Unified Quality Measures for Clusterings, Layouts, and Orderings of Graphs, and Their Application as Software Design Criteria

Von der Fakultät für Mathematik, Naturwissenschaften und Informatik
der Brandenburgischen Technischen Universität Cottbus

zur Erlangung des akademischen Grades

Doktor der Naturwissenschaften (Dr. rer. nat.)

genehmigte Dissertation

vorgelegt von

Dipl.-Inf. Andreas Noack
geboren am 09. April 1976 in Lauchhammer

Gutachter: Prof. Dr. rer. nat. Claus Lewerentz
Gutachter: Prof. Dr. rer. nat. Gregor Engels
Gutachter: Prof. Dr.-Ing. Andreas Zeller

Tag der mündlichen Prüfung: 06. Juli 2007

Abstract

How good is a given graph clustering, graph layout, or graph ordering – specifically, how well does it group densely connected vertices and separate sparsely connected vertices? How good is a given software design – specifically, how well does it minimize the interdependence of the subsystems? This work introduces and validates simple and uniform measures for these two properties. Together with existing optimization algorithms, the introduced measures enable the automatic computation e.g. of communities in social networks and of design flaws in software systems.

The first part derives, validates, and unifies quality measures for graph clusterings, graph layouts, and graph orderings, with the following results:

- Identical quality measures can be applied to clusterings, layouts, and orderings; this enables the computation of consistent clusterings, layouts, and orderings.
- Diverse existing and new measures can be unified into few general measures; this facilitates their comparison and validation.
- Many existing measures are biased towards certain clusterings, layouts, or orderings, even for graphs without particularly dense or sparse subgraphs, and thus do not (only) measure quality in the above sense.
- For example graphs, the minimization of new, unbiased (or weakly biased) measures reveals nonobvious groups, e.g. communities in social networks, subject areas in hypertexts, or closely interlocked countries in international trade.

The second part derives, validates, and unifies dependency-based indicators of software design quality. It applies two quality measures for graph clusterings as measures for the coupling of software subsystems – specifically for the coupling indicated by common changes and for the coupling indicated by references – and shows:

- The measures quantify the dependency-caused development costs, under well-defined simplifying assumptions.
- The minimization of the measures conforms to existing dependency-related design principles (like locality of change, acyclicity of references, and stability of references), design rules, and design patterns.
- For example software systems, the incremental minimization of the measures reveals nonobvious design flaws, like the distribution of coherent responsibilities over several subsystems, or references from low-level to high-level subsystems.

In summary, this work shows that

- simple measures can suffice to capture important aspects of graph clustering quality, graph layout quality, graph ordering quality, and software design quality, and
- the optimization of simple measures can suffice to detect nonobvious and often useful structure in various real-world systems.

Contents

List of Figures

List of Tables

Part I

Quality Measures for Graph Assignments

Chapter 1

Introduction

1.1 Goal

Surprising results can be obtained by applying some established clustering and layout algorithms to a trade network of seven European and three American countries: Not Europe is separated from America, but Sweden is separated from the rest of the world. Indeed, this separation is optimal – but with respect to the wrong criteria.[1]

This work is about criteria for identifying closely interlocked countries in international trade, groups of friends in social networks, subject areas in hypertexts, and cohesive modules in software systems; about criteria for identifying what researchers from Herbert Simon [Sim62] to Mark Newman [New03] have observed to be ubiquitous in real-world systems: weakly interacting groups of strongly interacting elements. Part I introduces, validates, and unifies quality measures for groupings, i.e. measures that quantify to what degree a given grouping clusters strongly interacting system elements and separates weakly interacting system elements. Part II, which is motivated in Chapter 9, applies these measures to evaluate design quality and identify design problems in software systems. Readers without interest in software engineering may skip Part II, missing a major application and some specific validation of the measures; readers without interest in data analysis may skip Part I, missing the general derivation and justification of the measures.

In this work, systems are modeled as *graphs*, with the system elements as vertices, the size of the system elements as vertex weights, the interactions between system elements as edges, and the interactions strengths as edge weights. Groupings of system elements are represented as clusterings, orderings, and layouts of graphs. A *clustering* of a graph partitions the set of vertices into disjoint subsets called clusters; an *ordering* imposes a total order on the vertices, and thereby assigns a rank to each vertex; and a *layout* assigns a position in low-dimensional Euclidean space to each vertex. Vertices are *grouped* in these representations if they belong to the same cluster, have similar ranks, or have a small Euclidean distance, respectively, and are *separated* otherwise. Then, in mathematical terms, the goal of Part I are simple and

[1] The trade network and the criteria will be described later in this section, after the necessary terminology has been introduced.

general *quality measures* for graph clusterings, graph orderings, and graph layouts, where quality is the degree to which densely connected vertices are grouped and sparsely connected vertices are separated.[2] Achieving this goal requires *unification*, to obtain simple and general measures, and *validation* (including the derivation of new valid measures), to ensure that measures actually measure quality in the specified sense; these two subgoals are detailed in the following two subsections.

1.1.1 Unification

Goal Concerning unification, the general goal is to transform different quality measures into similar mathematical expressions, in order to identify their commonalities and to unify them into more general (classes of) measures. A specific goal is to unify graph clusterings, graph orderings, and graph layouts into a more general class of graph representations (called *graph assignments*), to enable the unification of the respective quality measures.

Motivation The optimization of *identical* quality measures for clusterings, orderings, and layouts of a given graph enables the computation of *consistent* representations of the graph as clustering, ordering, and layout, which often reflect density and sparsity more precisely and more understandably than a single representation.

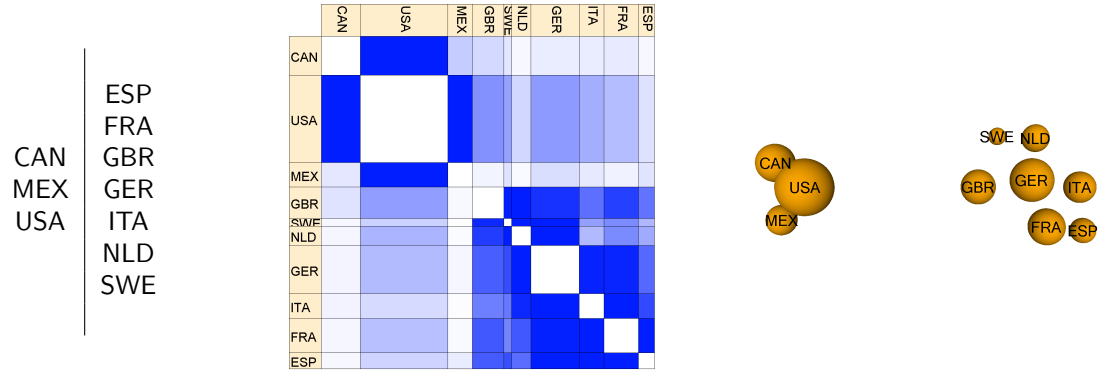
The transformation of measures into similar mathematical expressions facilitates their comparison and validation. Comparability is particularly important for users who need to select appropriate measures for specific applications. Validation benefits from the possibility to generalize results about one measure to similar measures.

Beyond practical problems, the current diversity and disorganization of measures shows that the property of grouping densely connected and separating sparsely connected vertices is not well understood; unification improves this understanding.

Example The running example in this chapter is a model of the trade between three North American and seven European countries. The vertices of the graph correspond to countries, and the edge weights specify the trade volume between each pair of countries. Because of geographical closeness and free trade agreements, the countries on the same continent trade more intensively than countries on different continents. More information about the graph can be found in Appendix A.1.

Figure 1.1 shows representations of the graph as clustering, ordering, and layout. Actually, the central subfigure shows not only an ordering, but a matrix visualization of the graph, where each vertex is visualized as a row and a column, and each edge weight is visualized as darkness of a matrix element. The graph ordering determines the order of the rows and the columns in the matrix.

[2] In this work, the term *quality* (of clusterings, orderings, and layouts) is used only in this specific sense. Other useful notions of quality exist, but there appears to be no established terminology to distinguish between these notions. In the literature, quality measures are also denoted as quality indexes, quality criteria, (internal) validity indexes, force models or energy models (for layouts), and objective functions, score functions, or fitness functions (in the context of optimization).



(a) Clustering (b) Ordering (c) Layout

Figure 1.1: Consistent representations of the American-European Trade graph

The clustering, the ordering, and the layout are consistent, as they all group countries from the same continent, and separate countries from different continents. This consistency is valuable, because all three representations offer complementary insights: The clustering makes the two groups of heavily trading countries explicit. The layout shows more details, for example that North America trades more intensively with GBR than with the other European countries. The ordering is less expressive, but enables a matrix visualization which (in contrast to matrices with arbitrarily ordered rows and columns) clearly shows the two groups, and many details about the intensity of trade within and between the groups.

Existing Results Identical quality measures for clusterings, orderings, and layouts are well-known in the analysis of proximity data, in particular when the input data are dissimilarity matrices (e.g. [CP80]), but graphs differ significantly from dissimilarity data (see Subsection 1.3.2). For graph clusterings, graph orderings, and graph layouts, similar algorithms with uniform intermediate results have been proposed (e.g. eigenvector-based algorithms), but not identical quality measures. Section 3.2 discusses previous work on uniform quality measures in more detail.

1.1.2 Validation

Goal Concerning validation, the main goals are to derive unbiased quality measures for graph clusterings, graph orderings, and graph layouts, and to detect biases (or their absence) in existing quality measures.

A quality measure for graph clusterings (or graph orderings, or graph layouts) is called *unbiased* if it takes the same value for all clusterings (or orderings, or layouts) of all graphs with uniform density. For directed graphs with loops (i.e. edges from a vertex to itself), the *density* of a subgraph is the total weight of its edges, divided by the squared total weight of its vertices. The adaption of this definition to undirected graphs and graphs without loops is straightforward (see Section 2.2).

Many graph models do not specify vertex weights explicitly. In this case, two common choices are to weight each vertex with 1, or to weight each vertex with its number of endvertices (i.e. its degree, the total weight of its incident edges). Accordingly, the notion of density splits into the density assuming unit vertex weights (shortly density$^{\text{vert}}$) and the density assuming unit endvertex weights (shortly density$^{\text{endv}}$) for graphs without vertex weights; the notion of (absence of) bias splits similarly.

Motivation In this work, a quality measure for graph assignments (i.e. for graph clusterings, graph orderings, or graph layouts) is considered to be valid if it quantifies the degree to which densely connected vertices are grouped and sparsely connected vertices are separated. Absence of bias is a *necessary* condition for validity: For a graph with uniform density, i.e. without densely connected or sparsely connected vertices, each graph assignment groups densely connected and separates sparsely connected vertices equally well. However, absence of bias is not *sufficient*; for example, every constant function is unbiased. The emphasis on the absence of bias, despite this limitation, is justified by its effectiveness in explaining failures of existing measures on certain graphs, and in the derivation of improved measures.

Many existing quality measures for graph assignments quantify a combination of several properties, like the number or the size of the clusters in a clustering, the scaling or the uniformity of the vertex distances in a layout, or the degree to which densely connected vertices are grouped and sparsely connected vertices are separated. It will be shown that the latter property alone is sufficient and even superior for some applications. But even if several properties need to be combined, it is still essential to know which properties a given measure quantifies, and thus the notion of bias and techniques for its detection can still be useful.

Example Figures 1.2, 1.3, and 1.4 show assignments of the American-European Trade graph described in the previous subsection. Each assignment was obtained by optimizing a quality measure for assignments of graphs without vertex weights.

The quality measure for Figure 1.2a is biased towards clusterings with one small and one large cluster, which explains the obtained clustering. The quality measure for Figure 1.4a is biased towards layouts with uniform vertex distances, and thus the resulting layout hardly shows any grouping.

The quality measures for Figures 1.2b, 1.3a, and 1.4b are unbiased$^{\text{vert}}$.[3] Such quality measures tend to assign vertices with small degree (i.e. small total weight of the incident edges) to small clusters or peripheral positions. Accordingly, the three assignments separate SWE from the remaining vertices.

The quality measures for Figures 1.2c, 1.3b, and 1.4c are unbiased$^{\text{endv}}$, and indeed group the countries according to their continents.

[3] To be precise, Figures 1.4b and 1.4c were produced with a small bias towards uniform vertex distances, to obtain more readable layouts (see Section 6.1). Generally, the goal is not to always avoid biases, but to know and control them.

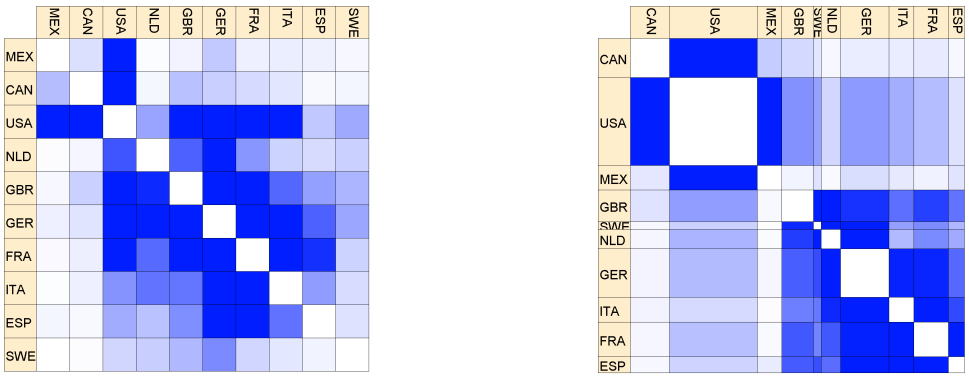
CAN		CAN			ESP
ESP		ESP			FRA
FRA		FRA			GBR
GBR		GBR		CAN	GER
GER	SWE	GER	SWE	MEX	ITA
ITA		ITA		USA	NLD
MEX		MEX			SWE
NLD		NLD			
USA		USA			

(a) Cut, total atedge length

(b) Ratio of the cut, normalized$^{\text{vert}}$ atedge length

(c) Shi-Malik normalized cut, normalized$^{\text{endv}}$ atedge length

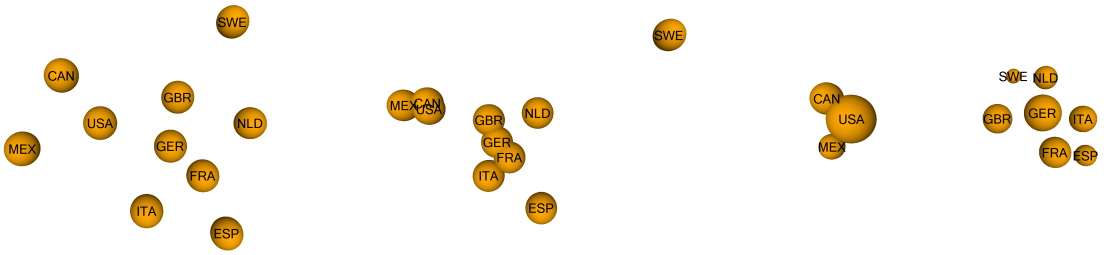
Figure 1.2: Clusterings of the American-European Trade graph



(a) Normalized$^{\text{vert}}$ atedge length

(b) Normalized$^{\text{endv}}$ atedge length

Figure 1.3: Orderings of the American-European Trade graph



(a) Fruchterman-Reingold (b) Normalized$^{\text{vert}}$ atedge length (c) Normalized$^{\text{endv}}$ atedge length

Figure 1.4: Layouts of the American-European Trade graph

Existing Results Most literature on quality measures for graph assignments does not consider vertex weights, and focuses on one type of graph assignment. A detailed discussion of existing quality measures for graph clusterings, graph orderings, and graph layouts can be found in Section 4.3, Section 5.2, and Section 6.4, respectively.

A simple quality measure for graph clusterings is the total weight of the inter-cluster edges, which is denoted as cut. Even if the single-cluster solution is excluded, the cut is biased towards clusterings with one small and one large cluster. Many cut-based measures with biases have been proposed, but also some unbiased variants. In particular, the ratio of the cut [LR88, MS90] is unbiased$^{\text{vert}}$, and Shi and Malik's normalized cut [SM00] and Newman's modularity [New04a] are unbiased$^{\text{endv}}$. Figure 1.2 shows clusterings with optimal cut, optimal ratio of the cut, and optimal Shi-Malik cut, respectively. Similarly to the checks for biases in this work, some authors have validated individual clustering quality measures by applying them to random graphs with uniform expected density [WC89, DHZ$^+$01, New06].

The most common quality measures for graph orderings are generalized sums of the edge lengths, which are unbiased$^{\text{vert}}$ (Figure 1.3a). Unbiased$^{\text{endv}}$ ordering quality measures are a contribution of this work (Figure 1.3b).

Quality measures for graph layouts are often denoted as force models or energy models. The prominent force and energy models of Eades [Ead84], Fruchterman and Reingold [FR91], and Davidson and Harel [DH96] are all strongly biased towards uniform edge lengths or uniform vertex distances (Figure 1.4a). Unbiased$^{\text{vert}}$ and unbiased$^{\text{endv}}$ energy models – more precisely, energy models with controllable bias – are a contribution of this work (Figure 1.4b and c).

1.2 Structure

Chapter 2 formally introduces the basic concepts of this work, in particular graphs, graph assignments (including graph clusterings, graph orderings, and graph layouts), and graph visualizations.

Chapter 3 derives a uniform and unbiased quality measure for graph assignments, and illustrates that the optimization of this measure produced consistent clusterings, orderings, and layouts.

Chapters 4, 5 and 6 concern graph clusterings, graph orderings, and graph layouts, and have a similar structure. They show that the quality measure of Chapter 3 can be transformed into a similar form as existing quality measures for the respective type of representation, which enables the application of existing algorithms for its optimization; they survey, unify, and validate (with respect to biases) existing quality measures; and they illustrate the benefits of unbiased measures for assignments of real-world graphs.

Chapter 7 introduces quality measures for a slightly modified notion of clustering quality, where vertices in different clusters are desired to be not connected at all, instead of sparsely connected (as in Chapters 3 and 4).

Chapter 8 concludes Part I with a summary and discussion of the main results.

1.3 Context and Limitations

This section provides context for the goals of Part I, and contrasts them with related non-goals.

1.3.1 From Data to Models

Various terms have been coined for the extraction of models (or summaries, or abstractions) from data. *Exploratory data analysis* typically employs visual and interactive techniques for identifying patterns in data; it can be followed by confirmatory data analysis, which checks how likely the identified patterns may have arisen purely by chance [Tuk77]. Similarly, statistics is commonly divided into *descriptive statistics*, which summarizes and visualizes data, and inferential or inductive statistics, which makes inferences about a population based on results obtained from samples [Man06, p. 3]. *Machine learning* searches hypotheses that fit observed data; as a research field, it differs from statistics in its stronger focus on algorithmic aspects [Mit97, Chapter 1]. *Data mining* primarily addresses large and sometimes noisy data sets [HMS01, Chapter 1]. *Knowledge discovery in data* denotes the entire process from the selection of the data to the interpretation of the extracted models, of which data mining is one specific step [FPSS96].

Automatic methods for the extraction of models from data have four basic components: A space of possible data sets, a space of possible models, a measure for the fit between models and data sets, and a method for searching models that optimize this measure (for a given data set). In this work, the data sets are graphs, and the models are graph assignments (including graph clusterings, graph orderings, and graph layouts). The goal is the derivation and validation of quality measures for graph assignments, *not* of search methods; suitable search methods for optimizing the new quality measures already exist.

1.3.2 The Data: Graphs vs. Proximities

This subsection clarifies the relationship between graph data, as used in this work, and two other common data formats, namely proximity data and vectorial data.

Proximity data is addressed in much of the literature on clusterings [JD88, KR90, AHS96, Gor99, ELL01] and layouts [BG97, CC01]. It specifies pairwise *proximities* for a set of objects. Proximities are either *dissimilarities* or *similarities*; the more two objects resemble each other, the smaller is their dissimilarity and the larger is their similarity.

One of the most common data formats, and in particular the basic data format of multivariate data analysis (e.g. [ED01, JW02]), is vectorial data. In *vectorial data*, each object (e.g. person) is described by the values of the same set of variables (e.g. age, gender, and weight), i.e. by a vector of values (if an ordering is imposed on the variables). Vectorial data can be transformed into proximity data using a *proximity measure*, i.e. a function that maps pairs of vectors to proximity values.

In this work, graphs have edge weights and optionally vertex weights, which are formally defined as multiplicities of edges and vertices (in Section 2.2), and are thus (at least) ratio-scaled[4]. Vertex weights have no counterpart in proximity data or vectorial data. Graphs without vertex weights can be transformed into ratio-scaled similarity data, and vice versa, by equating the edge weights with the similarities. Vectorial data may be indirectly transformed into graphs if a ratio-scaled similarity measure is available. While there are methods for transforming dissimilarities into similarities and vice versa (e.g. subtraction from a sufficiently large constant), there is no general method for transforming *ratio-scaled* dissimilarities into *ratio-scaled* similarities (and thus edge weights) or vice versa.

1.3.3 The Models: Clusterings, Orderings, and Layouts

The analysis or mining of real-world graphs is surveyed in a volume edited by Brandes and Erlebach [BE05] and several recent articles [Str01, AB02, New03, Wat04, CF06], and is partly covered in books on graph theory [Bol98, Die00, BJG01] and social network analysis [WF94, Sco00]. Clusterings, orderings, and layouts are among the most commonly used models in graph analysis, besides numerical descriptions of entire graphs or individual vertices and edges.

This work focuses on clusterings, orderings, and layouts, because they can naturally represent density and sparsity, by placing densely connected nodes densely (or in the same cluster) and sparsely connected nodes sparsely (or in different clusters). For layouts, there are even empirical studies showing that human viewers naturally interpret closely positioned vertices as strongly related [MBK97, DC98, HHE06]. The only model-related innovation of this work is the generalization of graph clusterings, graph orderings, and graph layouts to graph assignments.

1.3.4 Validation of Clusterings, Orderings, and Layouts

It is important to distinguish between the validity of quality measures for graph assignments (which is discussed in the next subsection), and the validity of graph assignments; the application of (valid) quality measures is one of several methods for validating graph assignments. This subsection is based on literature about clustering validity ([JD88, Chapter 4], [Gor99, Chapter 7], [ELL01, Chapter 8], [HBV01]), but is also applicable to other types of graph assignments.

Criteria The validity of a graph assignment has an internal and an external aspect. *Internal validity* requires the fit between the assignment and the graph. *External validity* requires the conformance of the assignment to a priori information, like an independently obtained assignment of the same graph.

Many different notions of internal validity have appeared in the literature. For example, clusterings are sometimes required to group vertices with similar neighbors,

[4]Introductions to scale types can be found in the literature on measurement theory ([Ste46],[Rob79, Chapter 2]) and software measurement ([FP96, Chapter 2], [Zus98, Chapter 4]).

as in the role assignments [Ler05] and blockmodels [NS05] of social network analysis, or vertices that are connected by short paths, as e.g. in k-cliques [WF94, Chapter 7], [Sco00, Chapter 6]. In automatic graph drawing, layouts are often optimized with respect to so-called aesthetic criteria, like a small number of edge crossings or a small variance of the edge lengths [DETT99], to produce readable visualizations of graphs.

This work focuses on a single, simple, widely studied, and practically useful notion of internal validity: Densely connected vertices should be grouped and sparsely connected vertices should be separated.

Methods The validation of graph assignments can be performed *qualitatively* or *quantitatively*. Internal validation verifies that densely connected vertices are grouped and sparsely connected vertices are separated, either qualitatively by inspecting visualizations of the graph and the assignment, or quantitatively by computing the value of an assignment quality measure (like the measures in this work). External validation compares the assignment with a given authoritative assignment, either qualitatively by inspecting visualizations of both assignments, or quantitatively by measuring the similarity of the assignments with an appropriate similarity measure.

Quantitative assessments may seem superior to qualitative assessments, but their appearance of objectivity and precision is often deceiving. The definition of an appropriate quality measure (for internal validity) or similarity measure (for external validity) may be much more difficult and error-prone than a direct qualitative assessment of assignments. Measures objectively validate the quality of assignments, but there is usually no objective validation of the quality of the measures. This work proposes and applies the identification of biases as an objective method for validating assignment quality measures.

Both the internal and the external validation of graph assignments may be either *relative* or *absolute*. Relative validation determines which of two assignments is better, and is performed, for example, by a clustering algorithm when it selects the best out of several computed clusterings. In quantitative assessments, it often boils down to a comparison of two measurement values. Absolute validation answers the question how "unusual" the graph assignment is under some null hypothesis. Typical null hypotheses are the randomness of the assignment, the randomness of the graph, or that the assignment was obtained by applying a specific method to a random graph. The degree of unusualness of an assignment under the null hypothesis can be determined by relating the actual value of a quality measure to the expected value of the quality measure under the null hypothesis. For example, the normalized atedge length, as main quality measure of this work, relates the total edge length of a given graph in a given assignment to the total edge length of a graph with uniform density in the same assignment. An alternative method, which will not be applied in this work, is to determine the degree of unusualness by statistical hypothesis testing, i.e. by computing the significance level at which the null hypothesis can be rejected.

To summarize, this work introduces measures for the internal validation of graph assignments, where internal validity means that densely connected vertices are grouped and sparsely connected vertices are separated.

1.3.5 Validation of Measures

Measures Measurement is the process of assigning numbers or symbols to attributes of objects. According to the representational theory of measurement ([Rob79], [FP96, Chapter 2], [Zus98, Chapter 4]), a *measure* is a mapping from an empirical relational system to a formal relational system. An *empirical relational system* consists of a set of objects, and relations between the objects that capture the intuitive understanding of the attribute of interest. A *formal relational system* consists of a set of symbols and relations between the symbols. For a quality measure for assignments of a graph G, the objects in the empirical relational system are the assignments of G. The main empirical relation is the binary relation "is better than" (denoted as $\prec$), where $p_1 \prec p_2$ if and only if p_1 groups densely connected vertices and separates sparsely connected vertices of G better than p_2. The symbols in the formal relational system are the real numbers, and the main formal relation is $<$ ("is smaller than"). An assignment quality measure for a graph G maps each assignment of G to a real number, and the empirical relation $\prec$ to the formal relation $<$.[5]

Internal Validity A measure is *internally valid* if it preserves the empirical relations. For example, an assignment quality measure Q for a graph G is internally valid if for all assignments p_1, p_2 of G holds: $p_1 \prec p_2$ if and only if $Q(p_1) < Q(p_2)$.

There are two complementary approaches to the internal validation of measures. Both approaches necessarily involve subjectivity, because the empirical relations are determined by human judgment. The *empirical* approach checks whether the empirical relations between a number of example objects conform to the formal relations between the corresponding measurement values. The *theoretical* approach proposes formal requirements for measures of a specific attribute, and verifies mathematically whether measures satisfy these requirements.

The empirical approach suffers from several limitations. It can only examine a limited number of objects, does not directly provide explanations and remedies for observed problems, and is relatively expensive. The efficiency of empirical studies is greatly improved if they are preceded by simple mathematical analyses that correct or weed out a significant number of invalid measures. Therefore, this work focuses on the theoretical approach, by defining the property of bias, analyzing assignment quality measures with respect to their biases, and deriving measures without bias. This approach will be demonstrated to be effective in explaining problems of existing measures and deriving improved measures, despite the fact that the absence of bias alone is not sufficient to ensure internal validity.

External Validity A measure is *externally valid* if it is related to an external attribute of the studied objects. Part II of this work will examine the external validity of specific assignment quality measures as indicators of software design quality. In Part I, the focus is on internal validity; however, evidence for the external validity

[5] Following the convention in most literature, better quality is reflected by *smaller* numbers.

of selected measures is provided by comparing assignments with small (i.e. good) measurement values to authoritative groupings derived from domain knowledge – e.g. by comparing the assignments in Figure 1.2 to 1.4 with the grouping of countries according to continents. Good, but not necessarily optimal, assignments are used for most measures because computing their global optima is infeasible for larger graphs, and only possibly suboptimal results of optimization heuristics are available. A fair comparison of different measures is ensured by using the same optimization heuristic for all compared measures.

To summarize, Part I of this work focuses on the theoretical evaluation of the internal validity of assignment quality measures, by checking the absence of bias. To a lesser extent, it also performs empirical evaluation of the internal and external validity of assignment quality measures, through qualitative internal and external validation of assignments obtained by a heuristic optimization of these measures.

1.4 Summary

Goals:

- uniform, unbiased quality measures for graph clusterings, graph orderings, and graph layouts
- identification of commonalities between, and biases in, existing quality measures for graph clusterings, graph orderings, and graph layouts
- where quality is defined as the degree to which densely connected vertices are grouped and sparsely connected vertices are separated

Non-goals:

- measures for other notions of assignment quality, like preservation of graph-theoretic distances, readability of visualizations (including classical graph drawing aesthetics), or conformance to given authoritative assignments
- quality measures for assignments of non-graph data, like dissimilarity data or vectorial data
- algorithms for optimizing quality measures (proven heuristics already exist), results about the time and space complexity of optimization problems
- extensive empirical evaluation of quality measures

Chapter 2

Basic Definitions

2.1 Sets and Multisets

For a set S, let $|S|$ be the number of elements of S, and let $S^{(k)}$ be the set of all subsets of S that have exactly k (distinct) elements.

A *multiset* $\mathcal{S}$ is a pair (S, m), where S is a set called the *underlying set*, and $\mathrm{m} : S \to \mathbb{N}_+$ is a function that assigns a positive *multiplicity* to each element of S. The domain of m will sometimes be implicitly extended beyond the underlying set S, such that $\mathrm{m}(s) = 0$ if $s \notin S$. Some multisets will have positive rational (instead of natural) multiplicities.

The notations for sets are naturally extended to multisets. In particular, $s \in \mathcal{S}$ if and only if $s \in S$; $\mathcal{S}$ is empty if and only if S is empty; the number of elements $|\mathcal{S}|$ is $\sum_{s \in S} \mathrm{m}(s)$; and sums over $\mathcal{S}$ contain $\mathrm{m}(s)$ terms for each $s \in S$, e.g. $\sum_{s \in \mathcal{S}} 1 = |\mathcal{S}|$. The multiset $\mathcal{S}^{\langle 2 \rangle}$ contains each set $\{s_1, s_2\} \subseteq S$ with multiplicity $\mathrm{m}(s_1)\mathrm{m}(s_2)$ if $s_1 \neq s_2$, and with multiplicity 0 if $s_1 = s_2$.

2.2 Graphs

A *graph* G is a pair $((V, \mathrm{w}), (E, \mathrm{f}))$ of nonempty multisets satisfying $E \subseteq V^{(2)}$. The elements of V are called *vertices*, the elements of E are called *edges*[1], and the functions w and f assign a *weight* to each vertex and edge, respectively. A vertex v is said to consist of $\mathrm{w}(v)$ *atvertices* (short for *atomic vertices*), (V, w) is called the *multiset of atvertices* of G, and similarly an edge e is said to consist of $\mathrm{f}(e)$ *atedges*.

An edge $\{u, v\}$ is said to *connect* the vertices u and v. The vertices u and v are *endvertices* of $\{u, v\}$. If $\{u, v\} \in E$, then the vertices u and v are *adjacent* or *neighbors* in G, and both u and v are *incident* with the edge $\{u, v\}$. The *degree* of a vertex $v \in V$ in G is the total weight of its incident edges, and is denoted as $\deg_G(v)$ or shortly $\deg(v)$. (Here as elsewhere the index referring to the graph is dropped

[1]Thus edges are unordered vertex pairs, and graphs are undirected. However, all concepts can be easily adapted to directed graphs (where edges are ordered vertex pairs), basically by replacing each unordered vertex pair with the two corresponding ordered vertex pairs.

if the reference is clear.) A vertex v is said to consist of $\deg(v)$ *endvertices*, and $(V, \deg)$ is called the *multiset of endvertices* of G.

The *density* or *intra-density* $\operatorname{den}(U)$ of a subset of vertices $U \subseteq V$ in G is the number of atedges within this subset, divided by the number of atvertex pairs, formally $\frac{|(U^{(2)},\mathrm{f})|}{|(U,\mathrm{w})^{\langle 2\rangle}|}$. Similarly, the *inter-density* $\operatorname{den}(V_1, V_2)$ of two disjoint subsets of vertices $V_1, V_2 \subseteq V$ in G is $\frac{|(V_1 \otimes V_2,\mathrm{f})|}{|(V_1,\mathrm{w})| \cdot |(V_2,\mathrm{w})|}$, where $V_1 \otimes V_2$ contains all sets with exactly one element from V_1 and one element from V_2.

A *subgraph* of G is a graph $((V', \mathrm{w}'), (E', \mathrm{f}'))$ where $V' \subseteq V$, $E' = E \cap V'^{(2)}$, and w' and f' are w and f restricted to the domain V' and E', respectively. Each subgraph of G is uniquely determined by, and is often identified with, its set of vertices.

A graph has *unit vertex weights* if each vertex has the weight 1, *unit endvertex weights* if each vertex v has the weight $\deg(v)$, and *unit edge weights* if each edge has the weight 1. Note that in graphs with unit endvertex weights, the multiset of atvertices equals the multiset of endvertices.

A *loop* is an element of $V^{(1)}$, i.e. connects a vertex with itself. In the computation of the degree of a vertex v, the weight of the loop $\{v\}$ is counted twice. The above definition of the intra-density changes to $\frac{|(U^{(1)} \cup U^{(2)},\mathrm{f})|}{\frac{1}{2}|(U,\mathrm{w})|^2}$ for graphs with loops. In this work, graphs have no loops except when loops are allowed explicitly.

2.3 Graph Assignments

A graph assignment is a model of a graph. The first subsection introduces graph assignments formally as mappings of the vertices of a graph to positions with certain distances. The next subsections define graph clusterings, graph orderings, and graph layouts as special graph assignments.

In this work, a graph assignment is considered as a *good* model of a graph if the distances between densely connected vertices are small, and the distances between sparsely connected vertices are large. This will be formalized in Chapter 3.

2.3.1 Graph Assignments

An *assignment* of a graph $G = ((V, \mathrm{w}), (E, \mathrm{f}))$ is a mapping from its set of vertices V to a metric space, i.e. to a set with an associated distance measure. A *distance measure* d on a set P is a mapping from pairs of elements of P to real numbers, such that for all $p_1, p_2, p_3 \in P$ holds

- $\mathrm{d}(p_1, p_2) \geq 0$ (nonnegativity),
- $\mathrm{d}(p_1, p_2) = 0$ if and only if $p_1 = p_2$ (identity),
- $\mathrm{d}(p_1, p_2) = \mathrm{d}(p_2, p_1)$ (symmetry), and
- $\mathrm{d}(p_1, p_2) \leq \mathrm{d}(p_1, p_3) + \mathrm{d}(p_3, p_2)$ (triangle inequality).

Different metric spaces will be used for different kinds of assignments, as defined in the following subsections. For an assignment p and two vertices $v_1, v_2 \in V$, the value $\mathrm{p}(v_1)$ is called the *position* of the vertex v_1 in p, and $\mathrm{d}(\mathrm{p}(v_1), \mathrm{p}(v_2))$ is called the distance of v_1 and v_2 in p.

2.3.2 Graph Clusterings and Cluster Graphs

A graph assignment is denoted as *graph clustering* if it maps the vertices to a metric space with the following distance measure:

$$\overline{\delta}(p_1, p_2) := \begin{cases} 0 & \text{if } p_1 = p_2 \\ 1 & \text{otherwise.} \end{cases}$$

The name $\overline{\delta}$ is chosen because $\overline{\delta}(p_1, p_2) = 1 - \delta(p_1, p_2)$, where δ is Kronecker's delta function. (Alternative distance measures are discussed in Section 4.2.) In clusterings, the positions are also called *clusters*.

A clustering p of a graph $G = ((V, \mathrm{w}), (E, \mathrm{f}))$ induces a mapping of each edge $\{u, v\} \in E$ to an edge $\{\mathrm{p}(u), \mathrm{p}(v)\}$ which is also denoted as $\mathrm{p}(\{u, v\})$. The set of resulting edges is denoted as $\mathrm{p}(E)$, and its elements are called *cluster edges* of G in p. The clustering p also induces a vertex weight function (denoted as $\mathrm{p}(\mathrm{w})$) which assigns to each cluster v' the sum $\sum_{v \in \mathrm{p}^{-1}(v')} \mathrm{w}(v)$ of the weights of the vertices that are mapped to v' by p, and similarly for edges. Thus the clustering p induces a graph $((\mathrm{p}(V), \mathrm{p}(\mathrm{w})), (\mathrm{p}(E), \mathrm{p}(\mathrm{f})))$ which is called *cluster graph* of G in p, and denoted as $\mathrm{p}(G)$.[2] Note that $\mathrm{p}(G)$ may have loops, even if G has no loops. To distinguish the cluster graph $\mathrm{p}(G)$ from G, the cluster vertices from vertices of G, and the cluster edges from edges of G, the latter are also called *base graph*, *base vertices*, and *base edges*, respectively, in the context of a clustering.

The rationale behind these definitions is that a clustering induces a summarized or aggregated or abstracted representation of the base graph: It maps the base graph to another graph called cluster graph, such that clusters of vertices of the base graph correspond to vertices of the cluster graph with the same weight, and clusters of edges of the base graph correspond to edges of the cluster graph with the same weight.

Related Work: Hierarchically Clustered Graphs Cluster graphs often arise as so-called views of hierarchically cluster*ed* graphs. Because of their practical relevance and the significant recent research interest [SZG+96, EH00, BW00, Rai02, vHvW04, AvH04, AKY05, NL05], the connection to these concepts is sketched here. A *hierarchically clustered graph* (G, T) (not to be confused with a cluster graph) consists of a graph G (called underlying graph) and a rooted tree T (called cluster tree), such that the leaves of T are exactly the vertices of G. Each vertex v of T corresponds to a subset of vertices of G which contains the leaf descendants of v. A subset V of vertices of T induces a clustering of G if each vertex of G is a leaf descendant of exactly one vertex in V. Cluster graphs of G in such clusterings are called *views* of the hierarchically clustered graph (G, T). As two extreme examples, the set of leaves of T induces a view which equals G (up to vertex names), and the singleton set containing the root of T induces a view with a single vertex.

[2] The unweighted variant of the cluster graph is often called quotient graph in the literature, e.g. [BC01b].

2.3.3 Graph Layouts

A *d-dimensional layout* p (for $d \in \mathbb{N}$, $d \geq 1$) of a graph $G = ((V, \mathrm{w}), (E, \mathrm{f}))$ is an assignment of G where the positions are vectors of d real numbers (i.e. $\mathrm{p}(V) \subseteq \mathbb{R}^{\mathrm{d}}$), and the distance measure is the Euclidean distance. The *Euclidean distance* of two positions $p_1, p_2 \in \mathbb{R}^{\mathrm{d}}$ is defined as

$$\|p_1 - p_2\| \ := \ \sqrt{\sum_{i=1}^{d} (p_{1,i} - p_{2,i})^2} \ ,$$

where $p_{1,i}$ and $p_{2,i}$ denote the ith element of the vector p_1 and p_2 respectively.

2.3.4 Graph Orderings

Graph orderings are special one-dimensional graph layouts with restricted vertex positions. The term ordering is chosen because an ordering of a graph is similar to a total ordering of its vertices. The restrictions of the vertex positions are motivated by a representation of the vertices as contiguous line segments, where the length of each line segment is the weight of the corresponding vertex. The position of each vertex in an ordering corresponds to the center of the corresponding line segment in such a representation (see Figure 2.1).

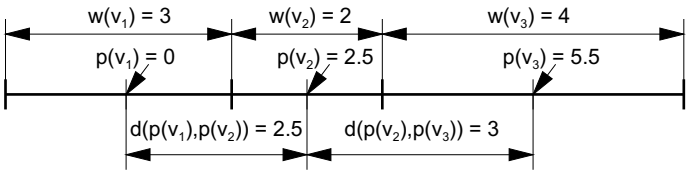


Figure 2.1: Positions and distances in an ordering p of a graph with three vertices

Formally, an assignment p of a graph $G = ((V, \mathrm{w}), (E, \mathrm{f}))$ is an *ordering* of G if all positions are real numbers, and there is a sequence $v_1, v_2, ..., v_{|V|}$ of the vertices in V with $\mathrm{p}(v_1) = 0$ and $\mathrm{p}(v_{i+1}) = \mathrm{p}(v_i) + \frac{1}{2}\mathrm{w}(v_i) + \frac{1}{2}\mathrm{w}(v_{i+1})$ for $i = 1, ..., |V|-1$. (Thus the set of positions is $\{0, 1, ..., |V|-1\}$ if the graph has unit vertex weights.)

Like layouts, orderings are always used in conjunction with the Euclidean distance as distance measure. Because the positions are real numbers, the Euclidean distance of two positions p_1 and p_2 is simply $|p_1 - p_2|$.

2.4 Graph Visualizations

In this work, graphs are visually represented as matrices and box-line diagrams. Basically, box-line diagrams represent vertices as points and edges as lines, while matrices represent vertices as lines and edges as points [Ber01]. Their readability in various graph analysis tasks is empirically compared in [GFC04, KEC06, NH01].

The primary rationale in the design of the visualizations is not beauty, but a clear and undistorted representation of the relevant graph properties. For example,

the proportionality of graph properties to the corresponding visual properties is not sacrificed only for a more uniform information density[3].

2.4.1 Matrix Visualizations

Representation of a Graph In a matrix visualization, a graph is represented as a two-dimensional matrix as follows:

- Each vertex is represented by exactly one row and one column of the matrix. The order of the vertices is the same for rows (from top to bottom) and columns (from left to right).
- The weight of each vertex equals the height of the corresponding row and the width of the corresponding column.
- Each unordered vertex pair, and in particular each edge, is represented by the two off-diagonal matrix elements where the row and the column of the vertices intersect. (Loops are represented by one matrix element at the diagonal.)
- The weight of each edge equals the color weight in each of the two corresponding matrix elements. (As an exception, the weight of each loop equals half of the color weight in the single corresponding matrix element.) The *color weight* of a matrix element is the product of its area and its color density. The *area* of a matrix element is, as usual, the product of its width and its height. The *color density* of a matrix element is its darkness in gray-scale figures, and its saturation in color figures. In both cases, white corresponds to a color density of 0. Intuitively, the color density is the density of the color pigments and the color weight is the total weight of the color pigments in the matrix element.

The *total area* of a set of matrix elements is the sum of the areas of its members, and the *total color weight* is defined similarly. The *average color density* of a set of matrix elements is the quotient of its total color weight and its total area. Note that the inter-density of a pair of different vertices equals the color density of the two corresponding matrix elements, and the inter-density of a pair of disjoint vertex sets equals the average color density of the corresponding matrix elements.

The widths and heights of the matrix elements are comparable within each matrix visualization, but not necessarily between different matrix visualizations, because the visualizations are scaled to fit the available space. Similarly, the color density in each matrix visualization is scaled such that most non-white matrix elements have neither an extremely small nor an extremely large darkness or saturation.

The problem of how easily and precisely visual attributes like the color weight are perceived by human viewers is beyond the scope this work. The chosen representation facilitates the comparison of the color density of matrix elements (and thus of inter-densities), rather than the comparison of the color weight (and thus of edge weights).

[3] In his classic book on the visual display of quantitative information, Tufte introduces the proportionality of numbers to their visual representations as a principle of graphical integrity, and coins the term *lie factor* for the degree of deviation from this principle [Tuf83, Chapter 2].

Representation of a Graph Clustering A graph clustering is represented by assigning the vertices of each cluster to contiguous rows and columns, and by separating vertices of different clusters with thick lines between the corresponding rows and columns.

Representation of a Graph Ordering A graph ordering is represented by a matrix visualization, if the distance of the vertices in the ordering equals the geometrical distance of the corresponding row centers (and column centers) in the visualization, and the vertex with the position 0 corresponds to the top row (and the left column) of the visualization. The definition of graph orderings restricts the positions such that there are no gaps or overlaps of rows or columns in the representing matrix visualizations.

Representation of a Graph Layout Graph layouts (except the special case of graph orderings) will not be represented in matrix visualization.

Example Figure 2.2a shows a matrix visualization of a graph $G = ((V, \mathrm{w}), (E, \mathrm{f}))$ with

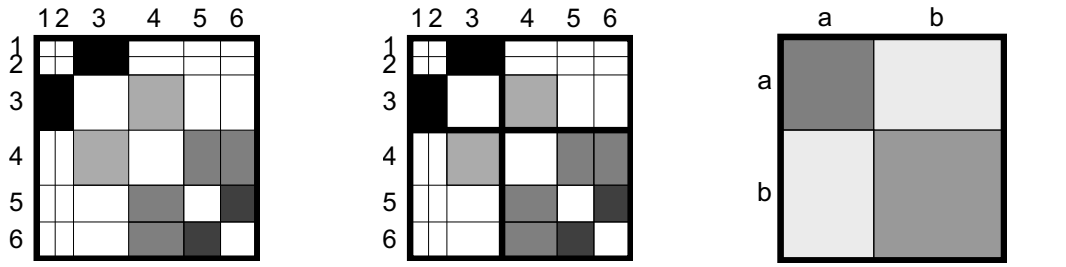
- the set of vertices $V = \{1, 2, 3, 4, 5, 6\}$,
- vertex weights $\mathrm{w}(1) = \mathrm{w}(2) = 1$, $\mathrm{w}(3) = \mathrm{w}(4) = 3$, $\mathrm{w}(5) = \mathrm{w}(6) = 2$,
- the set of edges $E = \{\{1, 3\}, \{2, 3\}, \{3, 4\}, \{4, 5\}, \{4, 6\}, \{5, 6\}\}$, and
- the edge weight 1 for all edges.

Note that the graph G has unit endvertex weights and unit edge weights.

Figure 2.2a also represents an ordering p of the graph G with $\mathrm{p}(1) = 0$, $\mathrm{p}(2) = 1$, $\mathrm{p}(3) = 3$, $\mathrm{p}(4) = 6$, $\mathrm{p}(5) = 8.5$, $\mathrm{p}(6) = 10.5$.

Figure 2.2b shows the graph G with a clustering p, with $\mathrm{p}(1) = \mathrm{p}(2) = \mathrm{p}(3)$ and $\mathrm{p}(4) = \mathrm{p}(5) = \mathrm{p}(6)$. The names of the clusters are not shown in the visualization; they do not affect the properties of clusterings that are examined in this work.

Figure 2.2c shows the cluster graph of G in p.



(a) A graph (b) A graph with a clustering (c) The resulting cluster graph

Figure 2.2: Matrix visualizations

2.4.2 Box-Line Visualizations

Representation of a Graph In a box-line visualization, a graph is represented in two-dimensional space as follows:

- Each vertex is represented as a filled circle.
- The weight of each vertex equals the area of the corresponding circle.
- Each edge is represented as a straight line that connects the representations of its end vertices. (As an exception, loops are represented as circular lines.) Sometimes the edges are not displayed to avoid clutter.
- The weight of each edge equals the width of the corresponding line.

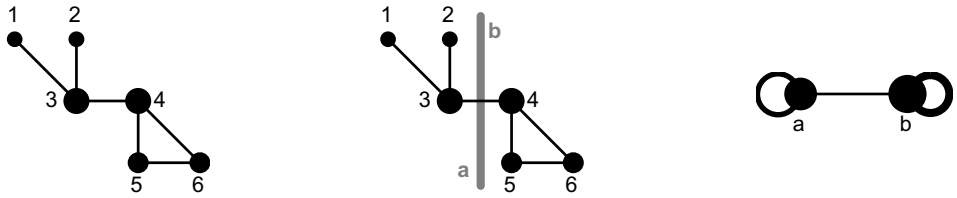
According to this definition, the "color weight" of a line equals the product of the edge length (i.e. the geometric distance of the representations of its endvertices) and the edge weight. It may appear more natural (and more compatible to the definition of the matrix visualization) to have a color weight proportional only to the edge weight. However, this is impractical because it leads to drastically varying line widths.

Representation of a Graph Clustering A graph clustering is represented by separating vertices of different clusters with fat gray lines.

Representation of a Graph Layout A one-dimensional or two-dimensional graph layout is represented by using the position vector of each vertex in the layout as coordinate vector of its representation.

Representation of a Graph Ordering Graph orderings can be treated as special graph layouts. However, orderings will mostly be visualized in matrix visualizations.

Example Figure 2.3 shows box-line visualizations of the graph, the clustering, and the cluster graph that were defined in the previous subsection and visualized as matrices in Figure 2.2.



(a) A graph (b) A graph with a clustering (c) The resulting cluster graph

Figure 2.3: Box-line visualizations

Besides the graph G, Figure 2.3a also shows a layout of G. The absolute positions of the vertices cannot be determined from the visualization because the coordinate system is not specified. However, the distances of the vertices can be determined from the visualization, and only these distances are relevant in this work.

2.5 Graphs without Vertex Weights

In the definition of graphs in Section 2.2, both vertices and edges have weights. However, most related literature concerns graphs without vertex weights, and many practical graph models do not specify vertex weights. A *graph without vertex weights* is a pair $(V, (E, \mathrm{f}))$, where V, E, and f are defined as for graphs.

A graph without vertex weights can be considered as a graph with implicit vertex weights, and can be transformed into a graph with vertex weights by making these weights explicit. Two natural and common (of many possible) weighting schemes for vertices are unit vertex weights and unit endvertex weights. Accordingly, each concept for graphs with vertex weights splits into two concepts for graphs without vertex weights, one assuming unit vertex weights (marked with $^{\mathrm{vert}}$) and one assuming unit endvertex weights (marked with $^{\mathrm{endv}}$). Of the concepts defined in this chapter, this concerns intra-density, inter-density, cluster graph, graph ordering, matrix visualization, and box-line visualization; the other concepts do not depend on the vertex weights.

For example, let $G = (V, (E, \mathrm{f}))$ be a graph without vertex weights, with

- the set of vertices $V = \{1, 2, 3, 4, 5, 6\}$,
- the set of edges $E = \{\{1, 3\}, \{2, 3\}, \{3, 4\}, \{4, 5\}, \{4, 6\}, \{5, 6\}\}$, and
- the edge weight 1 for all edges.

Then $\mathrm{den}^{\mathrm{vert}}(V) = \frac{6}{15}$, $\mathrm{den}^{\mathrm{endv}}(V) = \frac{6}{58}$, and Figure 2.4 shows a matrix$^{\mathrm{vert}}$ visualization (left) and a matrix$^{\mathrm{endv}}$ visualization (right) of G.

Not least, the quality of graph assignments is defined in terms of density, as the degree to which densely connected vertices are grouped and sparsely connected vertices are separated (see Section 1.1). Thus assignment quality depends on the vertex weights, and splits into assignment quality$^{\mathrm{vert}}$ and assignment quality$^{\mathrm{endv}}$ for graphs without vertex weights. For this reason, sections that introduce new quality measures for graph assignments have a subsection named "On Graphs without Vertex Weights", which compares the measures for these two notions of assignment quality.

Unlike graphs without vertex weights, graphs without edge weights are not discussed separately. All concepts for graphs with edge weights can be easily applied to graphs without edge weights by assuming unit edge weights.

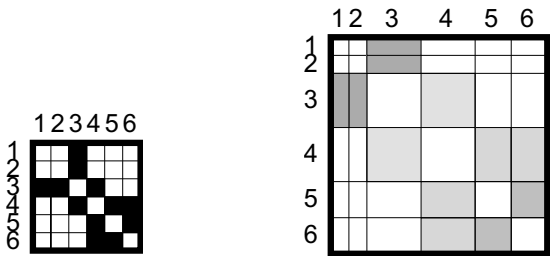


Figure 2.4: Matrix$^{\mathrm{vert}}$ and matrix$^{\mathrm{endv}}$ visualization of a graph without vertex weights

Chapter 3

Quality Measures for Graph Assignments

This chapter introduces uniform and unbiased quality measures for graph clusterings, graph orderings, and graph layouts. As defined in Section 1.1, these measures formalize that in a good graph assignment, densely connected vertices have small distances, and sparsely connected vertices have large distances.

The first section defines the quality measures and checks them for biases. The second section discusses related work on uniform quality measures. The final section demonstrates for real-world examples that the optimization of uniform quality measures indeed produces consistent clusterings, orderings, and layouts. Results and related work that are specific to either clusterings, orderings, or layouts are discussed in the following chapters.

3.1 Definitions and Analysis

This section derives quality measures for graph assignments by a stepwise identification and removal of biases, starting with the total atedge length as a very simple measure. For all quality measures, smaller values indicate better assignments.

3.1.1 The Total Atedge Length

The *total atedge length* of a graph $G = (\mathcal{V}, \mathcal{E})$ in an assignment p with respect to a distance measure d is

$$\mathrm{Q(p)} \;:=\; \sum\nolimits_{\{v_1,v_2\}\in\mathcal{E}} \mathrm{d(p}(v_1), \mathrm{p}(v_2)) \;.$$

The total atedge length rewards small distances between densely connected vertices, but not large distances between sparsely connected vertices. An assignment that places all vertices at the same position has the optimal total atedge length of 0. This bias towards assignments with small atvertex distances is formally identified in the next subsection, and removed in the following subsections.

3.1.2 Bias of the Total Atedge Length

As defined in Subsection 1.1.2, a quality measure for graph assignments is unbiased if it takes the same value for all assignments of all graphs with uniform density.

A graph $(\mathcal{V}, \mathcal{E})$ with $\mathcal{V} = (V, \mathrm{w})$ has the density $\frac{|\mathcal{E}|}{|\mathcal{V}^{\langle 2 \rangle}|}$. If the graph has uniform density, then each vertex pair $\{v_1, v_2\} \in V^{(2)}$ has the same inter-density, and is thus connected with an edge of weight $\frac{|\mathcal{E}|}{|\mathcal{V}^{\langle 2 \rangle}|}\mathrm{w}(v_1)\mathrm{w}(v_2)$. Therefore a nonempty multiset of atvertices $\mathcal{V}$ and a number of atedges $m \in \mathbb{N}_+$ uniquely determine a graph with uniform density, which is denoted as $\mathcal{G}(\mathcal{V}, m)$. On the other hand, each graph $(\mathcal{V}, \mathcal{E})$ with uniform density equals $\mathcal{G}(\mathcal{V}, |\mathcal{E}|)$. Figures 3.1a and 3.1b show $\mathcal{G}(\mathcal{V}, 4)$ where $\mathcal{V}$ contains one vertex of weight 2 and two vertices of weight 1.

Now the biases of the total atedge length can be determined by computing the total atedge length in all assignments p of $\mathcal{G}(\mathcal{V}, m)$. The following theorem states this total atedge length is proportional to both the density of the graph $\frac{m}{|\mathcal{V}^{\langle 2 \rangle}|}$, and the total distance of all atvertices $\sum_{\{v_1,v_2\} \in \mathcal{V}^{\langle 2 \rangle}} \mathrm{d}(\mathrm{p}(v_1), \mathrm{p}(v_2))$, but depends on no other properties of the assignment.

Theorem 3.1 *Let* d *be a distance measure, let* $\mathcal{V}$ *be a nonempty multiset of atvertices, let* $m \in \mathbb{N}_+$*, and let* p *be an assignment of* $\mathcal{G}(\mathcal{V}, m)$*. Then the total atedge length* $\mathrm{Q}(\mathrm{p})$ *of* $\mathcal{G}(\mathcal{V}, m)$ *in* p *with respect to* d *is*

$$\frac{m}{|\mathcal{V}^{\langle 2 \rangle}|} \cdot \sum_{\{v_1,v_2\} \in \mathcal{V}^{\langle 2 \rangle}} \mathrm{d}(\mathrm{p}(v_1), \mathrm{p}(v_2)) \ .$$

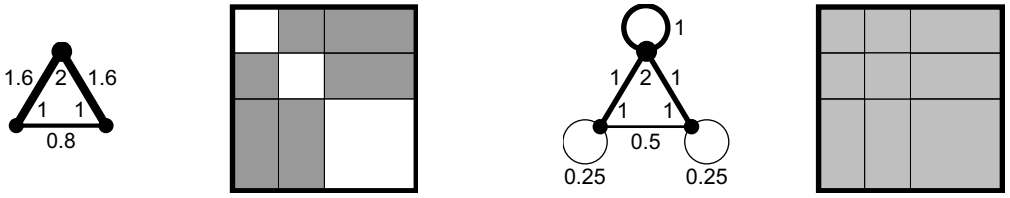
Proof: Let $\mathcal{G}(\mathcal{V}, m) = ((V, \mathrm{w}), (E, \mathrm{f}))$.

$$\begin{aligned}
\mathrm{Q}(\mathrm{p}) &= \sum_{\{v_1,v_2\} \in (E,\mathrm{f})} \mathrm{d}(\mathrm{p}(v_1), \mathrm{p}(v_2)) \\
&= \sum_{\{v_1,v_2\} \in E} \mathrm{f}(\{v_1, v_2\}) \mathrm{d}(\mathrm{p}(v_1), \mathrm{p}(v_2)) \\
&= \sum_{\{v_1,v_2\} \in V^{(2)}} \frac{|(E, \mathrm{f})|}{|\mathcal{V}^{\langle 2 \rangle}|} \mathrm{w}(v_1)\mathrm{w}(v_2) \mathrm{d}(\mathrm{p}(v_1), \mathrm{p}(v_2)) \\
&= \frac{m}{|\mathcal{V}^{\langle 2 \rangle}|} \cdot \sum_{\{v_1,v_2\} \in \mathcal{V}^{\langle 2 \rangle}} \mathrm{d}(\mathrm{p}(v_1), \mathrm{p}(v_2))
\end{aligned}$$

□

The adaption of this theorem to graphs with loops is straightforward: Two vertices $v_1, v_2 \in V$ in a graph with uniform density $(\mathcal{V} = (V, \mathrm{w}), \mathcal{E})$ are connected with an edge of weight $\frac{|\mathcal{E}|}{\frac{1}{2}|\mathcal{V}|^2}\mathrm{w}(v_1)\mathrm{w}(v_2)$ if $v_1 \neq v_2$, and $\frac{|\mathcal{E}|}{|\mathcal{V}|^2}\mathrm{w}(v_1)\mathrm{w}(v_2)$ if $v_1 = v_2$ (Figure 3.1c); the total atedge length in any assignment p is $\frac{|\mathcal{E}|}{\frac{1}{2}|\mathcal{V}|^2} \cdot \sum_{\{v_1,v_2\} \in \mathcal{V}^{\langle 2 \rangle}} \mathrm{d}(\mathrm{p}(v_1), \mathrm{p}(v_2))$.

Biases could also be identified for random graphs with uniform *expected* density, instead of fixed graphs with uniform density. A random graph (without loops) with the multiset of atvertices $\mathcal{V}$ can be obtained by selecting m atedges independently and with equal probability $\frac{1}{|\mathcal{V}^{\langle 2 \rangle}|}$ from $\mathcal{V}^{\langle 2 \rangle}$. (If a pair of vertices is selected k times, it is connected by one edge of weight k.) Then the expected edge weight between any unordered pair of vertices $\{v_1, v_2\} \in V^{(2)}$ is $\frac{m}{|\mathcal{V}^{\langle 2 \rangle}|}\mathrm{w}(v_1)\mathrm{w}(v_2)$, and the expected total atedge length of the random graph in any assignment p is the same as in Theorem 3.1.



(a) Without loops, box-line (b) Without loops, matrix (c) With loops, box-line (d) With loops, matrix

Figure 3.1: Graphs with uniform density. The numbers in the box-line visualizations specify the vertex weights and edge weights.

3.1.3 The Scaled Atedge Length

The scaled atedge length is obtained by normalizing the total atedge length with the total distance of all atvertices. Formally, the *scaled atedge length* of the graph $G = (\mathcal{V}, \mathcal{E})$ in the assignment p with respect to a distance measure d is

$$\mathrm{Q}^{\mathrm{scal}}(\mathrm{p}) \;:=\; \frac{\mathrm{Q}(\mathrm{p})}{\sum_{\{v_1,v_2\}\in\mathcal{V}^{\langle 2\rangle}} \mathrm{d}(\mathrm{p}(v_1),\mathrm{p}(v_2))} \;=\; \frac{\sum_{\{v_1,v_2\}\in\mathcal{E}} \mathrm{d}(\mathrm{p}(v_1),\mathrm{p}(v_2))}{\sum_{\{v_1,v_2\}\in\mathcal{V}^{\langle 2\rangle}} \mathrm{d}(\mathrm{p}(v_1),\mathrm{p}(v_2))} \;.$$

By Theorem 3.1, the scaled atedge length is $\frac{|\mathcal{E}|}{|\mathcal{V}^{\langle 2\rangle}|}$ for all assignments of all graphs $(\mathcal{V}, \mathcal{E})$ with uniform density.[1] Thus, the scaled atedge length is not biased towards any assignment for any graph with uniform density, but it is still biased towards graphs with small density.

3.1.4 The Normalized Atedge Length

The bias of the scaled atedge length towards graphs with small density can be removed by normalizing it with the density. Formally, the *normalized atedge length* of the graph $G = (\mathcal{V}, \mathcal{E})$ in the assignment p with respect to distance measure d is

$$\begin{aligned}\mathrm{Q}^{\mathrm{norm}}(\mathrm{p}) \;&:=\; \mathrm{Q}^{\mathrm{scal}}(\mathrm{p}) \,\Big/\, \frac{|\mathcal{E}|}{|\mathcal{V}^{\langle 2\rangle}|} \\ &=\; \frac{\sum_{\{v_1,v_2\}\in\mathcal{E}} \mathrm{d}(\mathrm{p}(v_1),\mathrm{p}(v_2))}{|\mathcal{E}|} \,\Big/\, \frac{\sum_{\{v_1,v_2\}\in\mathcal{V}^{\langle 2\rangle}} \mathrm{d}(\mathrm{p}(v_1),\mathrm{p}(v_2))}{|\mathcal{V}^{\langle 2\rangle}|} \;.\end{aligned}$$

The second of these two formulations has the simple interpretation that the average atedge length is normalized with the average atvertex distance.

By Theorem 3.1, the normalized atedge length is 1 for all assignments of all graphs with uniform density, and thus satisfies the definition of unbiased quality measures in Section 1.1.2.

[1] The scaled atedge length is undefined for degenerate assignments where all vertices have the same position. Its definition could be easily extended to map such assignments to the value $\frac{|\mathcal{E}|}{|\mathcal{V}^{\langle 2\rangle}|}$.

3.1.5 On Graphs without Vertex Weights

A graph without vertex weights can be considered as a graph with implicit unit vertex weights or as a graph with implicit unit endvertex weights (see Section 2.5). Thus assignment quality measures for graphs without vertex weights can be derived as special cases of the assignment quality measures for graphs with vertex weights. The resulting measures are listed in Subsection 3.4.3 on page 42.

By Theorem 3.1$^{\text{vert}}$, the normalized$^{\text{vert}}$ atedge length is 1 for all graphs with uniform density$^{\text{vert}}$, and is thus unbiased$^{\text{vert}}$. Theorem 3.1$^{\text{vert}}$ means the special case of Theorem 3.1 for graphs with unit vertex weights, i.e. for $\mathcal{G}(\mathcal{V}, m)$ where all vertices in $\mathcal{V}$ have weight 1. Similarly, by Theorem 3.1$^{\text{endv}}$ the normalized$^{\text{endv}}$ atedge length is 1 for all graphs with uniform density$^{\text{endv}}$, and is thus unbiased$^{\text{endv}}$. Theorem 3.1$^{\text{endv}}$ means Theorem 3.1 for graphs with unit endvertex weights, i.e. for $\mathcal{G}(\mathcal{V}, \frac{1}{2}|\mathcal{V}|)$.[2]

Uniform density$^{\text{vert}}$, without loops | Uniform density$^{\text{vert}}$, with loops | Uniform density$^{\text{endv}}$, without loops[3] | Uniform density$^{\text{endv}}$, with loops

Figure 3.2: Uniform-density graphs without vertex weights. The numbers specify the edge weights.

[2]If loops are *not* allowed, then all graphs with uniform density$^{\text{endv}}$ have uniform degrees: Suppose $(V, \mathcal{E})$ is a graph without vertex weights, without loops, and with the uniform density$^{\text{endv}}$ $d > 0$. Then the sum of the degrees of all vertices in V is $2|\mathcal{E}|$, and thus for each vertex $v \in V$ holds $\deg(v) = d \deg(v)(2|\mathcal{E}| - \deg(v))$, which implies $\deg(v) = 0$ or $\deg(v) = 2|\mathcal{E}| - \frac{1}{d}$.

In other words, there exists no uniform-density$^{\text{endv}}$ graph with a vertex set V and vertex degrees deg unless all vertices with nonzero degrees have the same degree. However, a graph with *approximately* uniform density$^{\text{endv}}$ can be constructed for any nonnegative vertex degrees deg by omitting the vertex weights from the uniform-density graph $\mathcal{G}\big((V, \deg), \frac{1}{2}|(V, \deg)|\big)$. This graph has the right set of vertices V, the right total edge weight $\frac{1}{2}|(V, \deg)|$, and approximately the right degrees deg, with a particularly good approximation if the vertex degrees are small (relative to the total edge weight) or uniform. If it had exactly the right degrees, it would have uniform density$^{\text{endv}}$ and its normalized$^{\text{endv}}$ atedge length would be 1 in all assignments.

Thus, if loops are not allowed, the normalized$^{\text{endv}}$ atedge length is not only 1 in assignments of graphs with uniform density$^{\text{endv}}$ (which exist only for particular vertex degrees), but also approximately 1 in assignments of the above-constructed graphs with approximately uniform density$^{\text{endv}}$ (which exist for any nonnegative vertex degrees).

If loops are allowed, a graph with uniform density$^{\text{endv}}$ can be constructed for any nonnegative vertex degrees deg by connecting each unordered vertex pair $\{v_1, v_2\} \in V^{(2)}$ with an edge of weight $\frac{1}{|(V,\deg)|} \deg(v_1) \deg(v_2)$ if $v_1 \neq v_2$, and $\frac{1}{2|(V,\deg)|} \deg(v_1) \deg(v_2)$ if $v_1 = v_2$.

[3]For reasons discussed in the previous footnote, this graph has only approximately uniform density$^{\text{endv}}$, with inter-densities of $\frac{0.8}{2.4 \cdot 2.4} \approx 0.139$ for the lower edge and $\frac{1.6}{3.2 \cdot 2.4} \approx 0.208$ for the upper edges. It has been constructed by omitting the vertex weights from $\mathcal{G}(\mathcal{V}, 4)$ where $\mathcal{V}$ contains two vertices of weight 2 and one vertex of weight 4. The approximation of uniform density$^{\text{endv}}$ is not very good because the vertex degrees are not much smaller than the total edge weight.

To understand the difference between the normalized$^{\text{vert}}$ atedge length and the normalized$^{\text{endv}}$ atedge length, it is helpful to calculate the value of the normalized$^{\text{vert}}$ atedge length for graphs with uniform density$^{\text{endv}}$, and the value of the normalized$^{\text{endv}}$ atedge length for graphs with uniform density$^{\text{vert}}$. These values are given in the following two theorems.

Theorem 3.2 states that for graphs with uniform density$^{\text{endv}}$, the normalized$^{\text{vert}}$ atedge length of an assignment is the quotient of the average endvertex distance and the average vertex distance. This quotient is 1 if all vertices have the same degree (i.e. the same number of endvertices), but otherwise, the quotient can be minimized by placing the vertices with the highest degree at central positions, i.e. at positions with small distances to the other vertices. Thus the normalized$^{\text{vert}}$ atedge length is always 1 for graphs with uniform density$^{\text{vert}}$ (by Theorem 3.1$^{\text{vert}}$), but prefers assignments with high-degree vertices at central positions for graphs with uniform density$^{\text{endv}}$ (by Theorem 3.2).

Theorem 3.2 *Let* d *be a distance measure, let* $\mathcal{V} = (V, \mathrm{w})$ *be a nonempty multiset of atvertices, and let* p *be an assignment of* $\mathcal{G}(\mathcal{V}, \frac{1}{2}|\mathcal{V}|)$*. Then the normalized*$^{\text{vert}}$ *atedge length* $\mathrm{Q}^{\text{normvert}}(\mathrm{p})$ *of* $\mathcal{G}(\mathcal{V}, \frac{1}{2}|\mathcal{V}|)$ *in* p *with respect to* d *is*

$$\frac{\sum_{\{v_1,v_2\}\in(V,\deg)^{\langle 2\rangle}} \mathrm{d}(\mathrm{p}(v_1),\mathrm{p}(v_2))}{|(V,\deg)^{\langle 2\rangle}|} \Big/ \frac{\sum_{\{v_1,v_2\}\in V^{(2)}} \mathrm{d}(\mathrm{p}(v_1),\mathrm{p}(v_2))}{|V^{(2)}|} \ .$$

Proof: According to Theorem 3.1,

$$\mathrm{Q}(\mathrm{p}) \ = \ \frac{\frac{1}{2}|\mathcal{V}|}{|\mathcal{V}^{\langle 2\rangle}|} \cdot \sum_{\{v_1,v_2\}\in\mathcal{V}^{\langle 2\rangle}} \mathrm{d}(\mathrm{p}(v_1),\mathrm{p}(v_2)) \ .$$

By the definition of $\mathrm{Q}^{\text{normvert}}$,

$$\begin{aligned}
\mathrm{Q}^{\text{normvert}}(\mathrm{p}) &= \frac{\mathrm{Q}(\mathrm{p})}{\sum_{\{v_1,v_2\}\in V^{(2)}} \mathrm{d}(\mathrm{p}(v_1),\mathrm{p}(v_2))} \Big/ \frac{\frac{1}{2}|\mathcal{V}|}{|V^{(2)}|} \\
&= \frac{\frac{1}{2}|\mathcal{V}|}{|\mathcal{V}^{\langle 2\rangle}|} \cdot \frac{\sum_{\{v_1,v_2\}\in\mathcal{V}^{\langle 2\rangle}} \mathrm{d}(\mathrm{p}(v_1),\mathrm{p}(v_2))}{\sum_{\{v_1,v_2\}\in V^{(2)}} \mathrm{d}(\mathrm{p}(v_1),\mathrm{p}(v_2))} \Big/ \frac{\frac{1}{2}|\mathcal{V}|}{|V^{(2)}|} \\
&= \frac{\sum_{\{v_1,v_2\}\in\mathcal{V}^{\langle 2\rangle}} \mathrm{d}(\mathrm{p}(v_1),\mathrm{p}(v_2))}{|\mathcal{V}^{\langle 2\rangle}|} \Big/ \frac{\sum_{\{v_1,v_2\}\in V^{(2)}} \mathrm{d}(\mathrm{p}(v_1),\mathrm{p}(v_2))}{|V^{(2)}|} \\
&= \frac{\sum_{\{v_1,v_2\}\in(V,\deg)^{\langle 2\rangle}} \mathrm{d}(\mathrm{p}(v_1),\mathrm{p}(v_2))}{|(V,\deg)^{\langle 2\rangle}|} \Big/ \frac{\sum_{\{v_1,v_2\}\in V^{(2)}} \mathrm{d}(\mathrm{p}(v_1),\mathrm{p}(v_2))}{|V^{(2)}|} \ .
\end{aligned}$$

□

Theorem 3.3 states that for graphs with uniform density$^{\text{vert}}$, the normalized$^{\text{endv}}$ atedge length of any assignment is 1. Thus the normalized$^{\text{endv}}$ atedge length is unbiased$^{\text{endv}}$ (by Theorem 3.1$^{\text{endv}}$) and unbiased$^{\text{vert}}$ (by Theorem 3.3).

Theorem 3.3 *Let* d *be a distance measure, let* (V, w) *be a nonempty multiset of atvertices with unit vertex weights, let* $m \in \mathbb{N}_+$*, and let* p *be an assignment of* $\mathcal{G}((V, \mathrm{w}), m)$*. Then the normalized*$^{\mathrm{endv}}$ *atedge length* $\mathrm{Q}^{\mathrm{normendv}}(\mathrm{p})$ *of* $\mathcal{G}((V, \mathrm{w}), m)$ *in* p *with respect to* d *is* 1*.*

Proof: According to Theorem 3.1,

$$\begin{aligned} \mathrm{Q}(\mathrm{p}) &= \frac{m}{|(V,\mathrm{w})^{\langle 2\rangle}|} \cdot \sum\nolimits_{\{v_1,v_2\}\in(V,\mathrm{w})^{\langle 2\rangle}} \mathrm{d}(\mathrm{p}(v_1),\mathrm{p}(v_2)) \\ &= \frac{m}{|V^{(2)}|} \cdot \sum\nolimits_{\{v_1,v_2\}\in V^{(2)}} \mathrm{d}(\mathrm{p}(v_1),\mathrm{p}(v_2)) \ . \end{aligned}$$

By the definition of $\mathrm{Q}^{\mathrm{normendv}}$,

$$\begin{aligned} \mathrm{Q}^{\mathrm{normendv}}(\mathrm{p}) &= \frac{\mathrm{Q}(\mathrm{p})}{\sum_{\{v_1,v_2\}\in(V,\deg)^{\langle 2\rangle}} \mathrm{d}(\mathrm{p}(v_1),\mathrm{p}(v_2))} \Big/ \frac{m}{|(V,\deg)^{\langle 2\rangle}|} \\ &= \frac{m}{|V^{(2)}|} \cdot \frac{\sum_{\{v_1,v_2\}\in V^{(2)}} \mathrm{d}(\mathrm{p}(v_1),\mathrm{p}(v_2))}{\sum_{\{v_1,v_2\}\in(V,\deg)^{\langle 2\rangle}} \mathrm{d}(\mathrm{p}(v_1),\mathrm{p}(v_2))} \Big/ \frac{m}{|(V,\deg)^{\langle 2\rangle}|} \\ &= \frac{|(V,\deg)^{\langle 2\rangle}|}{|V^{(2)}|} \cdot \frac{\sum_{\{v_1,v_2\}\in V^{(2)}} \mathrm{d}(\mathrm{p}(v_1),\mathrm{p}(v_2))}{\sum_{\{v_1,v_2\}\in V^{(2)}} \deg(v_1)\deg(v_2)\mathrm{d}(\mathrm{p}(v_1),\mathrm{p}(v_2))} \ . \end{aligned}$$

Because all vertices in (V, w) have the same weight 1, all vertices in $\mathcal{G}((V, \mathrm{w}), m)$ have the same degree $\frac{2m}{|V|}$, and $|(V, \deg)^{\langle 2\rangle}| = \left(\frac{2m}{|V|}\right)^2 |V^{(2)}|$. Thus

$$\mathrm{Q}^{\mathrm{normendv}}(\mathrm{p}) = \frac{|(V,\deg)^{\langle 2\rangle}|}{|V^{(2)}|} \cdot \left(\frac{|V|}{2m}\right)^2 \cdot \frac{\sum_{\{v_1,v_2\}\in V^{(2)}} \mathrm{d}(\mathrm{p}(v_1),\mathrm{p}(v_2))}{\sum_{\{v_1,v_2\}\in V^{(2)}} \mathrm{d}(\mathrm{p}(v_1),\mathrm{p}(v_2))} = 1 \ . \qquad \square$$

There is a notable asymmetry in Theorem 3.2 and Theorem 3.3: The normalized$^{\mathrm{vert}}$ atedge length is biased for graphs with uniform density$^{\mathrm{endv}}$, but the normalized$^{\mathrm{endv}}$ atedge length is not biased for graphs with uniform density$^{\mathrm{vert}}$. This is unsurprising, given that graphs with uniform density$^{\mathrm{vert}}$ also have uniform density$^{\mathrm{endv}}$, but not vice versa. The difference between the normalized$^{\mathrm{vert}}$ atedge length and the normalized$^{\mathrm{endv}}$ atedge length will be demonstrated on example clusterings, orderings, and layouts in the following chapters.

3.2 Related Work

The main contributions of this chapter are uniform quality measures for graph clusterings, graph orderings, and graph layouts, and the identification and removal of biases in these measures. The first subsection discusses related work in the analysis of proximity data, where the fitting of different kinds of models using a single quality measure has already received some attention. In graph analysis, uniform quality measures appear to be unknown, but so-called spectral methods derive clusterings, orderings, and layouts from a common intermediate representation, which is

outlined in the second subsection. The third subsection discusses bias-related evaluation techniques for assignment quality measures. Related work that is specific to graph clusterings, graph orderings, or graph layouts is surveyed in the Sections 4.3, 5.2, and 6.4, respectively.

3.2.1 Uniform Quality Measures in Proximity Analysis: Proximity Measures for Proximity Matrices

Uniform quality measures are enabled by the unification of graph clusterings, graph orderings, and graph layouts into a single class of graph representations. The graph assignments in Section 2.3 appear to be the first proposal of such a uniform representation in graph analysis.

However, the representation of clusterings as proximity matrices has been exploited for a long time in the analysis of proximity data (see e.g. the discussions of cophenetic proximities in [JD88, Section 3.2], [SC96]), and the representation of layouts as proximity matrices is obvious. If the input data are also proximity matrices, then proximity measures for (proximity) matrices can be used as quality measures for clusterings and layouts, as outlined in the first paragraph of this subsection. The total atedge length may be seen as proximity measure for matrices, and indeed has been used as such in proximity analysis under the name Mantel's Z, as discussed in the second paragraph. The problem of permuting the rows and columns of a matrix to minimize Mantel's Z is known as quadratic assignment problem. The final paragraph compares this problem with the problem of finding a graph assignment with minimal total atedge length.

Proximity Measures for (Proximity) Matrices Proximity matrices are a common input format of data analyses (see Section 1.3.2), and many analysis results, including clusterings and layouts, can be represented as proximity matrices (see Section 2.3, [JD88, Section 3.2], [SC96]). If both the input and the results are represented as proximity matrix, then the quality of the results may be determined with any proximity measure for matrices.

This approach is probably most convincing when the input and the output are either both similarity matrices or both dissimilarity matrices. Then a large similarity (or a small dissimilarity) of the output matrix to the input matrix means that the output is a good approximation of the input.

Methods for comparing matrices are surveyed in [AH96, Section 5]. In particular, any proximity measure for vectors can also be used as proximity measure for matrices, by considering an $n \times n$ matrix as a vector of length n^2 (or as a shorter vector when some matrix elements, for example along the diagonal, are trivial and therefore ignored). Particularly the Euclidean distance (or the squared Euclidean distance, which is often called least squares criterion) has been used as quality measure for various different results of data analysis, namely for layouts in unidimensional scaling ([BG97, Section 13.5], [CC01, Section 2.6]) and multidimensional scaling ([KW78,

Chapter 1], [BG97, Chapter 3]), for clusterings [SC96], and even for hybrid models combining clusterings and layouts [CP80].

Mantel's Z Mantel's Z is a similarity measure for matrices. It is equivalent to the total atedge length when it is applied to the adjacency matrix of a graph and the distance matrix of a graph assignment. Formally, for two $n \times n$ matrices M_1 and M_2, Mantel's Z is defined as

$$Z(M_1, M_2) \ := \ \sum_{i=1}^{n} \sum_{j=1}^{n} M_1(i,j) M_2(i,j) \ .$$

Mantel first used this measure to establish relationships between the temporal distances and the spatial distances of disease outbreaks [Man67]. Hubert and Schultz [HS76] and Hubert [Hub87] provide extensive treatments with many application examples. Both Mantel [Man67] and Hubert [Hub87, Section 4.2.2] derive the expectation (and also the variance) of the measure over all permutations of the rows and columns of M_1 (while Theorem 3.1 states the expectation of the total atedge length for random graphs, see the remarks below its proof). Based on this expectation, Hubert [Hub87, Sections 1.2.4, 4.2.2] proposes two normalizations, of which one (called $\mathcal{A}^{**}$) has a similar form as the normalized$^{\text{vert}}$ atedge length.

Quadratic Assignment Problem Mantel's Z is the quality measure for solutions of the well-known quadratic assignment problem (see [BcPP98, Cel98, Ans03] for surveys). In this problem, two $n \times n$ matrices M_1 and M_2 are given, and the goal is to permute the rows and columns of M_1 such that $Z(M_1, M_2)$ is minimized. For example, if M_1 specifies the flow of materials between n facilities, and M_2 specifies transportation cost (per unit of material) between n locations, then an optimal solution of the quadratic assignment problem assigns the facilities bijectively to the locations such that the total transportation costs are minimized. The quadratic assignment problem is considered as particularly hard, because even finding a solution within a constant factor of the optimum is NP-complete [SG76], and because even some moderately-sized problem instances (with around 30 vertices) have only recently been solved to optimality [Ans03].

In graph terminology, the goal of the quadratic assignment problem is to find a graph assignment with minimal total atedge length, with the restrictions that the set of vertex positions (with their pairwise distances) is given, and that only bijective mappings of the vertices to the positions are allowed. In other words, the problem of finding an optimal assignment of a given graph is generally not a quadratic assignment problem, because the positions in the resulting assignment are not fixed in advance, but chosen freely from a certain domain (like $\mathbb{R}^d$). In contrast to the quadratic assignment problem, the computation of a graph assignment with minimal total atedge length is trivial: Simply assign all vertices to the same position. In fact, the bias of the total atedge length towards assignments with small vertex distances was a main motivation for introducing more sophisticated measures, namely the scaled and the normalized atedge length.

3.2.2 Uniform Intermediate Representations in Graph Analysis: Spectral Methods

Spectral methods compute graph assignments from eigenvectors of matrix representations of graphs. They are popular mainly because both simple algorithms (like the power iteration, see e.g. [GL96, Chapter 8]) and efficient algorithms (like Lanczos methods [GL96, Chapter 9] and multi-scale algorithms [BS94, KCH03]) for computing eigenvectors are available. Each eigenvector of the matrix representation assigns a real number to each vertex of the graph, and thus each eigenvector is a one-dimensional graph layout. Clusterings are derived from these one-dimensional layouts e.g. by thresholding, orderings are derived by adjusting the spacing of the vertices, and nontrivial higher-dimensional layouts may be obtained by combining the one-dimensional layouts corresponding to several eigenvectors.

Spectral methods based on the so-called Laplacian matrix (defined below) are particularly popular for computing graph clusterings (e.g. [DH73, HK92, SM00]), graph orderings (e.g. [JM92, BPS95, ABH98, DH04]), and graph layouts (e.g. [Hal70, PST00, BN03, Kor05]). The remainder of this subsection will show that the resulting one-dimensional layouts minimize distance ratios similar to the scaled$^{\text{vert}}$ atedge length and the scaled$^{\text{endv}}$ atedge length (introduced in Subsection 3.1.5). However, there is no guarantee that the derived clusterings and orderings still minimize these distance ratios [JM92], and the respective higher-dimensional layouts are only solutions of more complex minimization problems with various constraints [BN03, Kor05].

Two common matrix representations of graphs without vertex weights are the adjacency matrix and the Laplacian. The *adjacency matrix* A of a graph without vertex weights $G = (V, (E, \mathrm{f}))$ is a symmetric $|V| \times |V|$ matrix with $A(v_1, v_2) := \mathrm{f}(\{v_1, v_2\})$ for all $v_1, v_2 \in V$. The *degree matrix* D of G is a the $|V| \times |V|$ diagonal matrix with $D(v, v) := \deg(v)$. The *Laplacian* L of G is defined as $L := D - A$. A number μ is called an *eigenvalue* of the Laplacian L with the associated *eigenvector* y if y is an non-zero vector with $Ly = \mu y$. For connected graphs, all eigenvalues of the Laplacian are real, the smallest eigenvalue is 0 (with associated eigenvector $(1, 1, ..., 1)^T$), and all other eigenvalues are positive [Fie75]. The eigenvector corresponding to the second smallest eigenvalue is called the *Fiedler vector*.

Theorem 3.4 (Fiedler [Fie75]) *The Fiedler vector of a graph without vertex weights* $(V, \mathcal{E})$ *minimizes*

$$\frac{\sum_{\{v_1,v_2\}\in\mathcal{E}} \mathrm{d}(\mathrm{p}(v_1), \mathrm{p}(v_2))^2}{\sum_{\{v_1,v_2\}\in V^{(2)}} \mathrm{d}(\mathrm{p}(v_1), \mathrm{p}(v_2))^2}$$

over all one-dimensional layouts p *with at least two different vertex positions.*

In the past decade, solutions of the generalized Laplacian eigensystem $Ly = \mu Dy$ have received considerable attention [Chu97, SM00, BN03, DH04, Kor05]. The generalized eigenvector corresponding to the second smallest generalized eigenvalue is called *endvertex-generalized Fiedler vector* in the following.

Theorem 3.5 (Chung [Chu97, Equation 1.5]) *The endvertex-generalized Fiedler vector of a graph without vertex weights* $(V, \mathcal{E})$ *minimizes*

$$\frac{\sum_{\{v_1,v_2\}\in\mathcal{E}} \mathrm{d}(\mathrm{p}(v_1), \mathrm{p}(v_2))^2}{\sum_{\{v_1,v_2\}\in(V,\deg)^{\langle 2\rangle}} \mathrm{d}(\mathrm{p}(v_1), \mathrm{p}(v_2))^2}$$

over all one-dimensional layouts p *with at least two different positions.*

There are two connections between Theorems 3.4 and 3.5 and the main results of this chapter. First, the distance ratio minimized by the Fiedler vector (Theorem 3.4) differs from the scaled$^{\text{vert}}$ atedge length only in the exponents of the distances in the numerator and the denominator. Thus both measures can be unified into a single measure with variable exponents; in fact, this generalized measure also subsumes many other existing measures, as the following chapters will show.

Second, the distinction between scaled$^{\text{vert}}$ (Theorem 3.4) and scaled$^{\text{endv}}$ (Theorem 3.5) quality measures has already been discovered (under different names) in the context of spectral methods. This distinction is crucial because scaled$^{\text{vert}}$ measures are biased towards assigning high-degree vertices to central positions (Theorem 3.2), while there is no such bias for scaled$^{\text{endv}}$ measures (Theorem 3.1$^{\text{endv}}$).

3.2.3 Theoretical Validation of Assignment Quality Measures

In Subsection 3.1.2, fixed graphs with uniform density and random graphs with uniform expected density have been used to identify biases of quality measures towards certain graph assignments. Similar random graph models (but for graphs without vertex weights) have already been used to validate clustering quality measures, as detailed in Subsection 4.3.6.

In Subsection 3.1.5, the bias towards assignments with high-degree vertices on central positions in the normalized$^{\text{vert}}$ atedge length (and its absence in the normalized$^{\text{endv}}$ atedge length) has been shown for graphs with uniform density$^{\text{endv}}$. (Alternatively, random graphs with uniform expected density$^{\text{endv}}$ could have been used.) This bias and its practical importance has already been observed empirically (e.g. in [SM00, DH04, Kor05]), but there appears to be no previous theoretical comparison of normalized$^{\text{vert}}$ and normalized$^{\text{endv}}$ measures with respect to their biases.

3.3 Examples

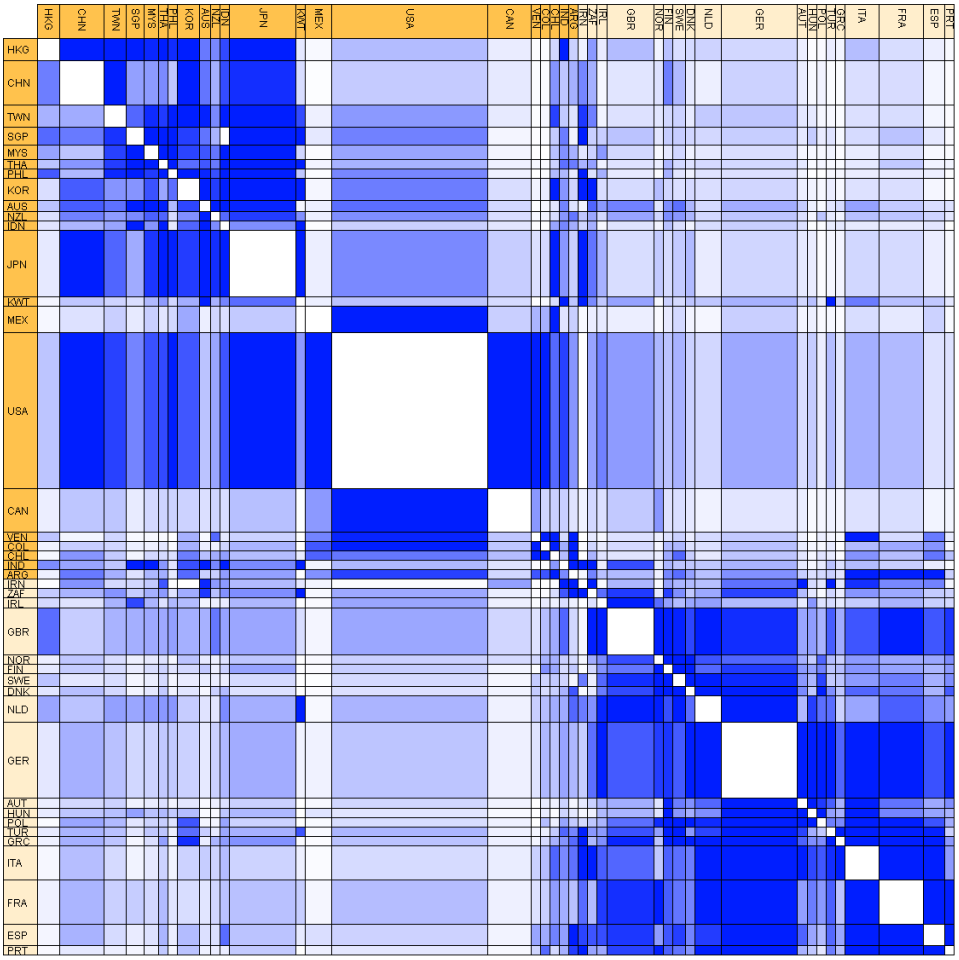
The two main practical motivations of introducing *uniform* quality measures for clusterings, orderings, and layouts were the computation of consistent representations of graphs, and a more efficient validation of the quality measures. The latter benefit was already exploited in Section 3.1, where the presence or absence of bias was not shown separately for clusterings, orderings, and layouts, but only once for each uniform quality measure. The purpose of this section is to illustrate the earlier benefit, i.e. the consistency of clusterings, orderings, and layouts, for some real-world graphs. The internal and external validity of the assignment quality measures is *not* addressed in this section, but in Sections 4.4, 5.3, and 6.5.

All graphs in this section are described in Appendix A.1. For each graph, a matrix visualization and a box-line visualization are presented. The matrix visualization shows a graph ordering, the box-line visualization shows a graph layout, and both show the same graph clustering. All graph clusterings have two clusters, which are indicated by the saturation (or darkness, in gray-scale printouts) of the vertex representations.

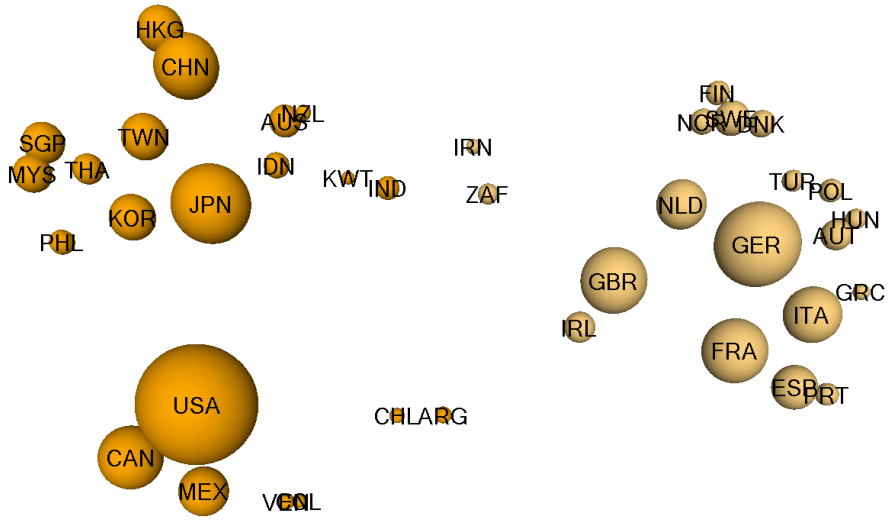
The clustering and ordering of each graph were obtained by minimizing the normalized$^{\text{endv}}$ atedge length. The layout was obtained by minimizing the 0-normalized$^{\text{endv}}$ atedge length (see Subsection 6.1), which also yields a layout with small (though generally not minimal) normalized$^{\text{endv}}$ atedge length. Because the exact minimization is algorithmically intractable, heuristics were used in all cases, and there is no guarantee that the presented assignments are optimal or at least near-optimal. (But there is evidence for near-optimality, in particular that similar final assignments were obtained for different random starting assignments.) Layouts were rotated manually to a similar orientation as the ordering; rotation does not change the normalized$^{\text{endv}}$ atedge length. Assignments with small normalized$^{\text{vert}}$ atedge length are not presented, as the consistency is similar as for assignments with small normalized$^{\text{endv}}$ atedge length.

The clusterings, orderings, and layouts are shown in Figure 3.3 to 3.9. The conclusions are similar for all graphs:

- The vertices of each cluster have contiguous positions in the ordering.
- The two clusters of the clustering correspond to the two most clearly separated groups of vertices in the layout. Some layouts (Morse Code Confusion, College Football) suggest several alternative splits of the vertex set into two clusters; additional measurements show that these alternative clusterings indeed have a similar normalized$^{\text{endv}}$ atedge length as the shown clustering.
- If the layout shows clearly separated groups of vertices, then the vertices of each group have contiguous positions in the ordering. If there is a dominant linearization of the vertices in the layout (e.g. West-East for US Airlines, plant-animal for Food Classification), then this linearization is consistent with the ordering.

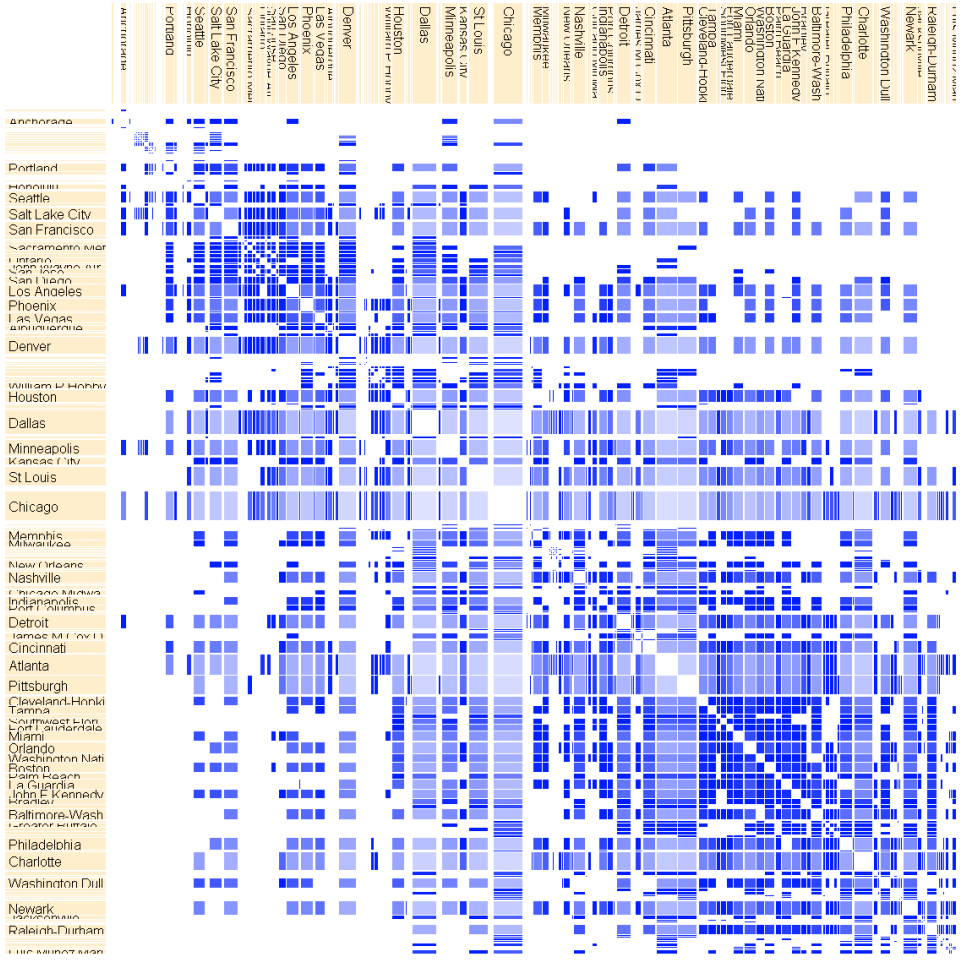


(a) Ordering with $Q^{\text{normendv}}(p) = 0.624$, and clustering with $Q^{\text{normendv}}(p) = 0.425$

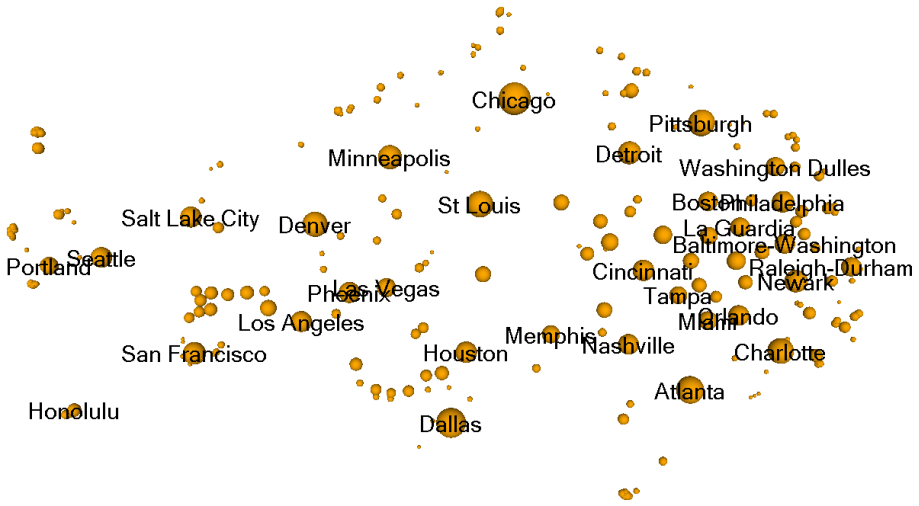


(b) Layout with $Q^{\text{normendv}}(p) = 0.598$, and clustering with $Q^{\text{normendv}}(p) = 0.425$

Figure 3.3: Assignments of the World Trade graph

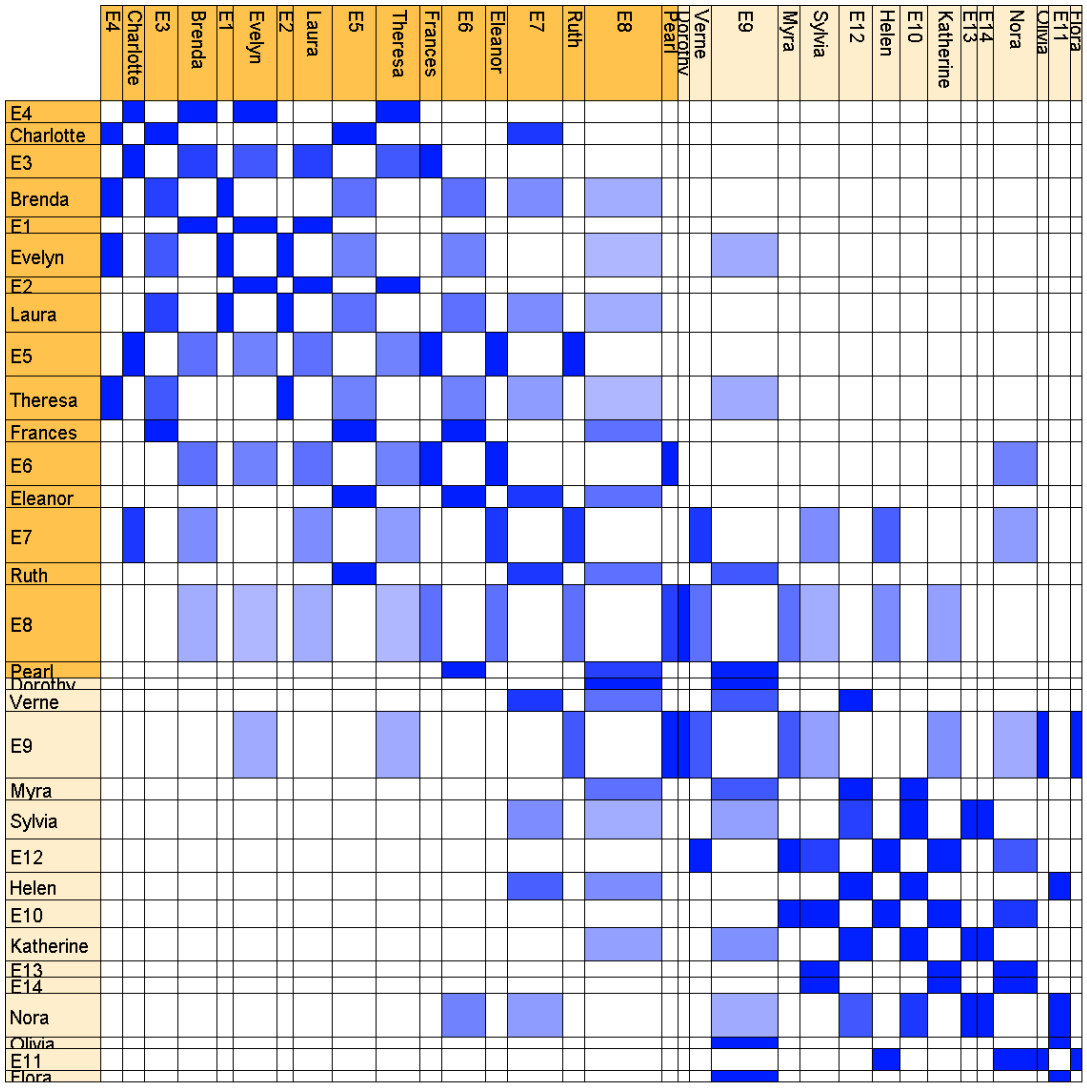


(a) Ordering with $Q^{\text{normendv}}(p) = 0.444$

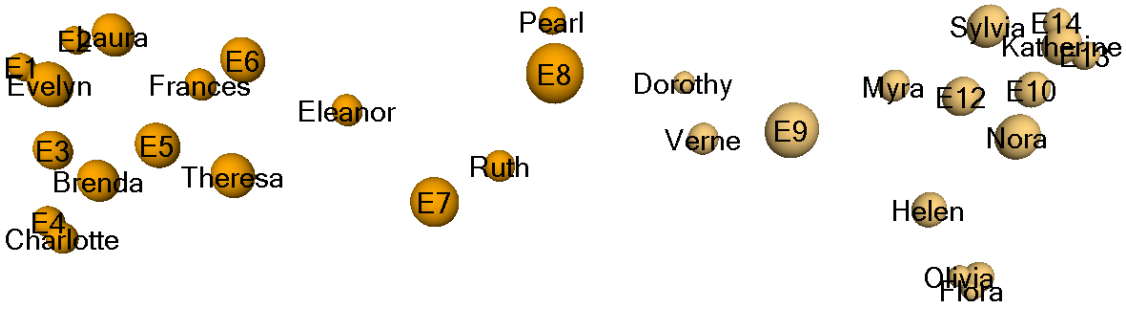


(b) Layout with $Q^{\text{normendv}}(p) = 0.419$. Some distant airports in Alaska and the South Sea (e.g. Guam) are omitted to improve readability.

Figure 3.4: Assignments of the US Airlines graph

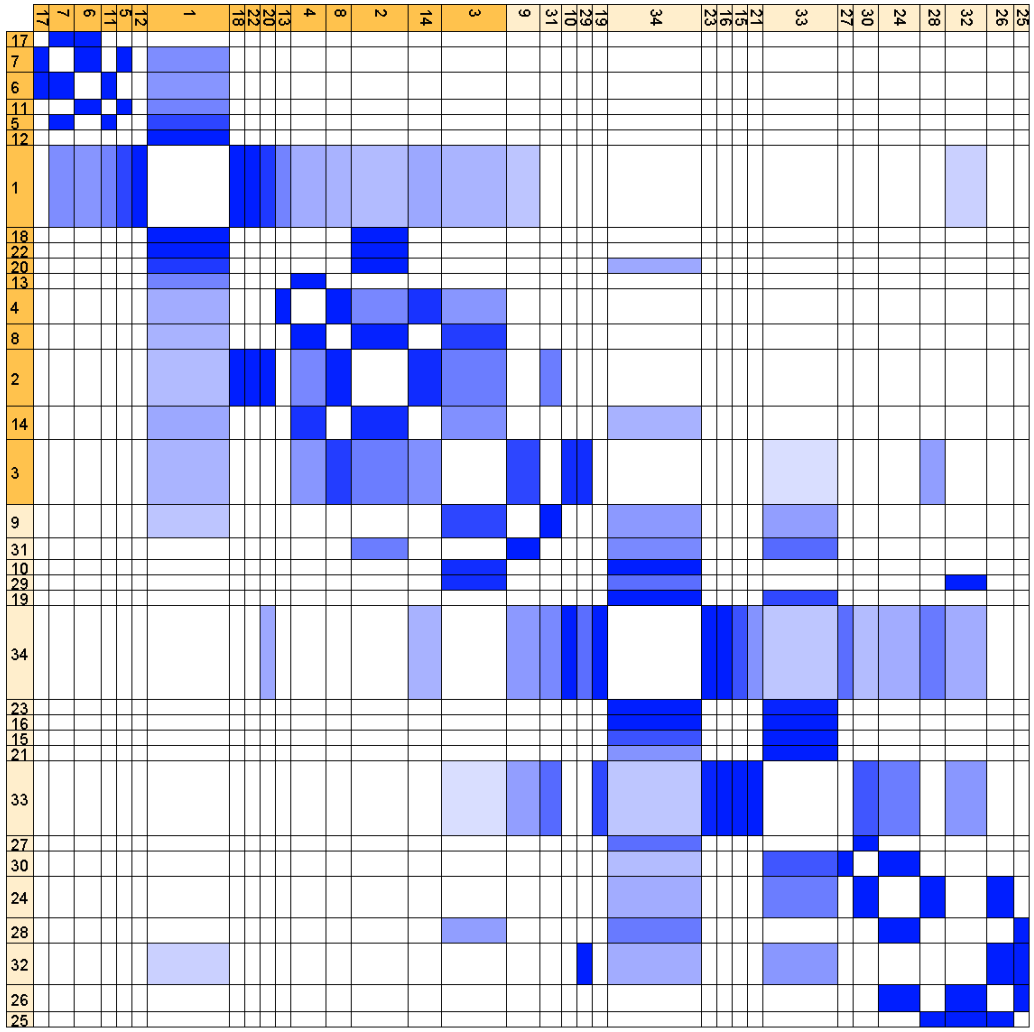


(a) Ordering with $Q^{\text{normendv}}(p) = 0.446$, and clustering with $Q^{\text{normendv}}(p) = 0.335$

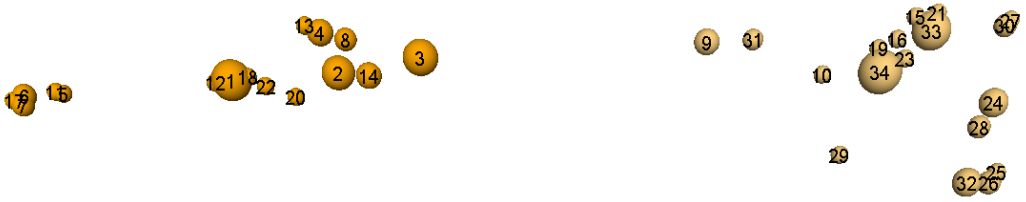


(b) Layout with $Q^{\text{normendv}}(p) = 0.430$, and clustering with $Q^{\text{normendv}}(p) = 0.335$

Figure 3.5: Assignments of the Southern Women graph

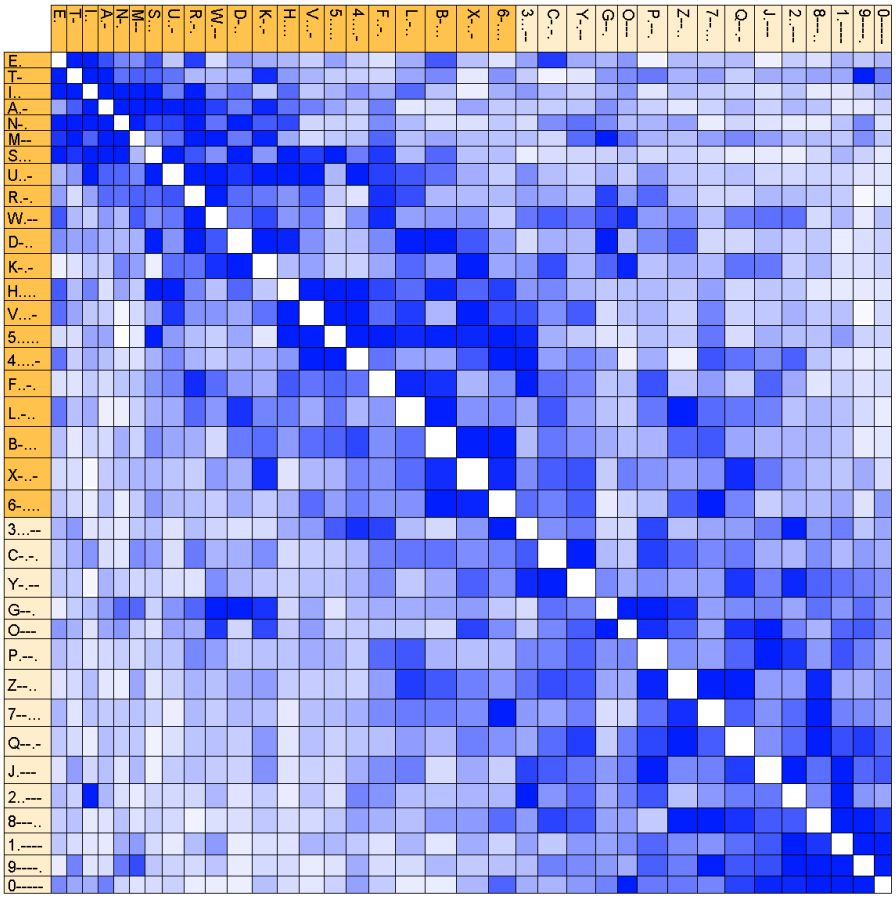


(a) Ordering with $Q^{\text{normendv}}(p) = 0.348$, and clustering with $Q^{\text{normendv}}(p) = 0.181$

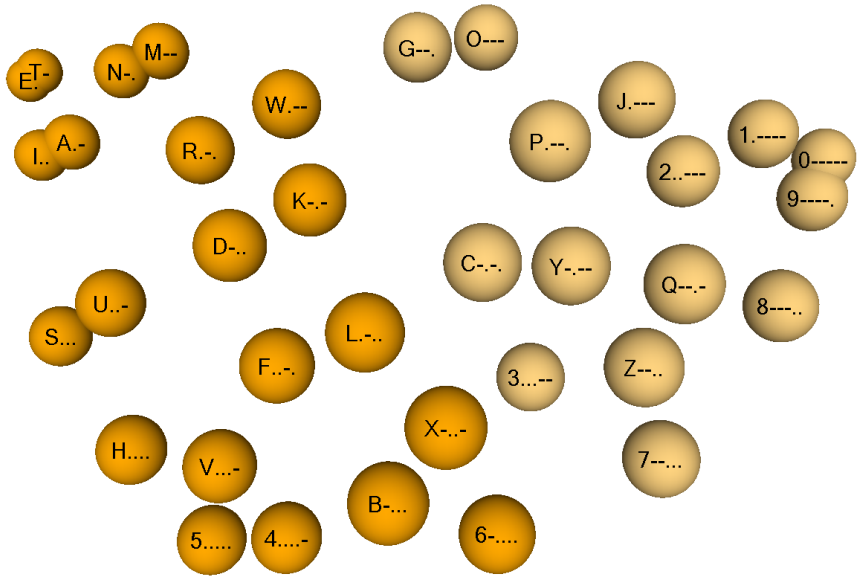


(b) Layout with $Q^{\text{normendv}}(p) = 0.276$, and clustering with $Q^{\text{normendv}}(p) = 0.181$

Figure 3.6: Assignments of the Karate Club graph

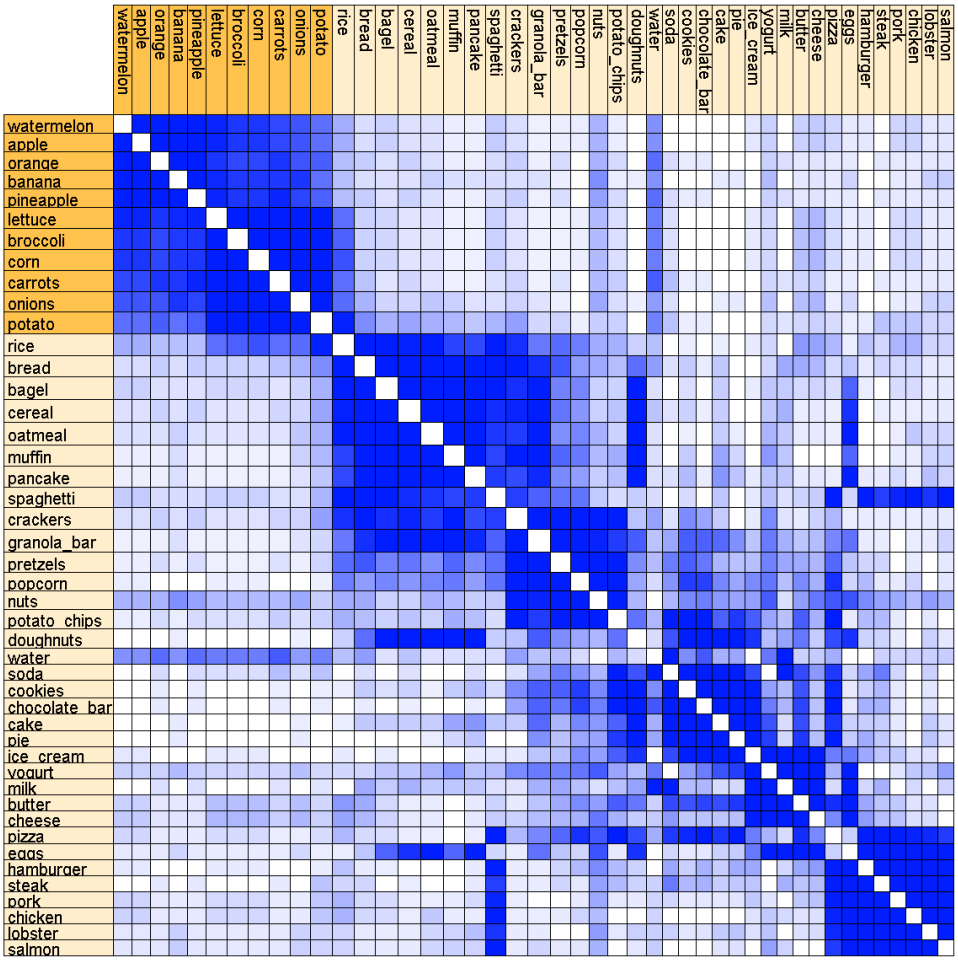


(a) Ordering with $Q^{\text{normendv}}(p) = 0.721$, and clustering with $Q^{\text{normendv}}(p) = 0.690$

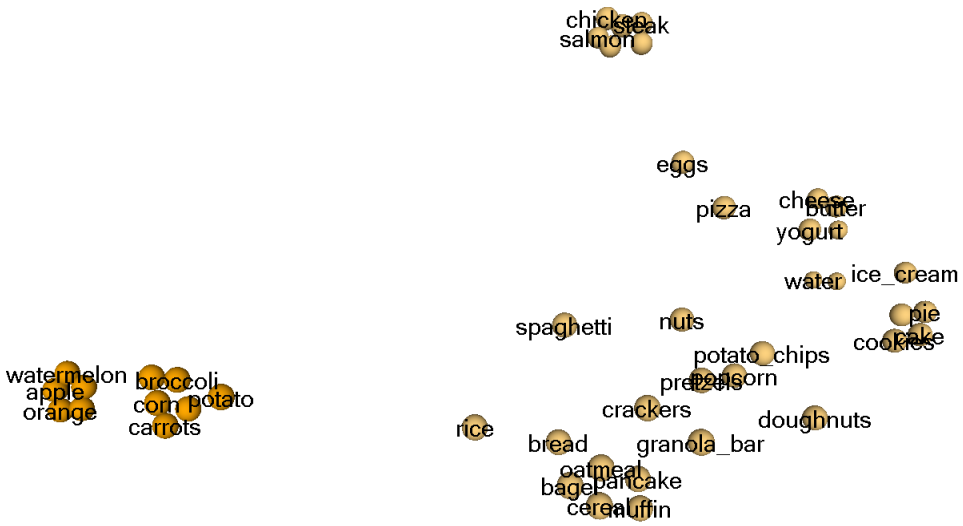


(b) Layout with $Q^{\text{normendv}}(p) = 0.765$, and clustering with $Q^{\text{normendv}}(p) = 0.690$

Figure 3.7: Assignments of the Morse Code Confusion graph

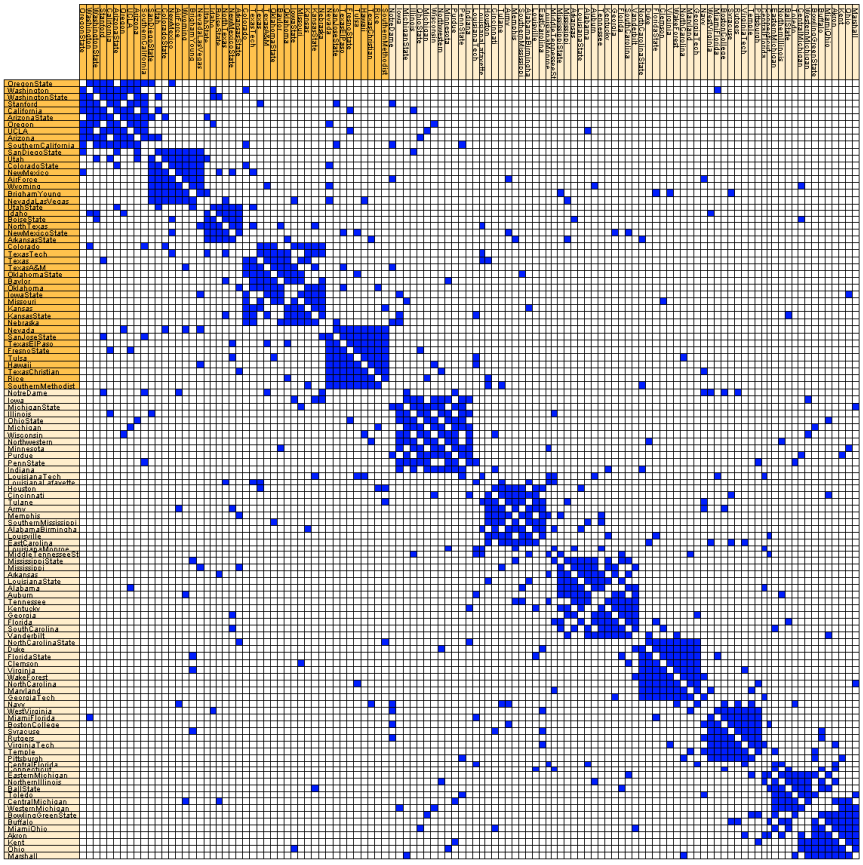


(a) Ordering with $Q^{\text{normendv}}(p) = 0.462$, and clustering with $Q^{\text{normendv}}(p) = 0.291$

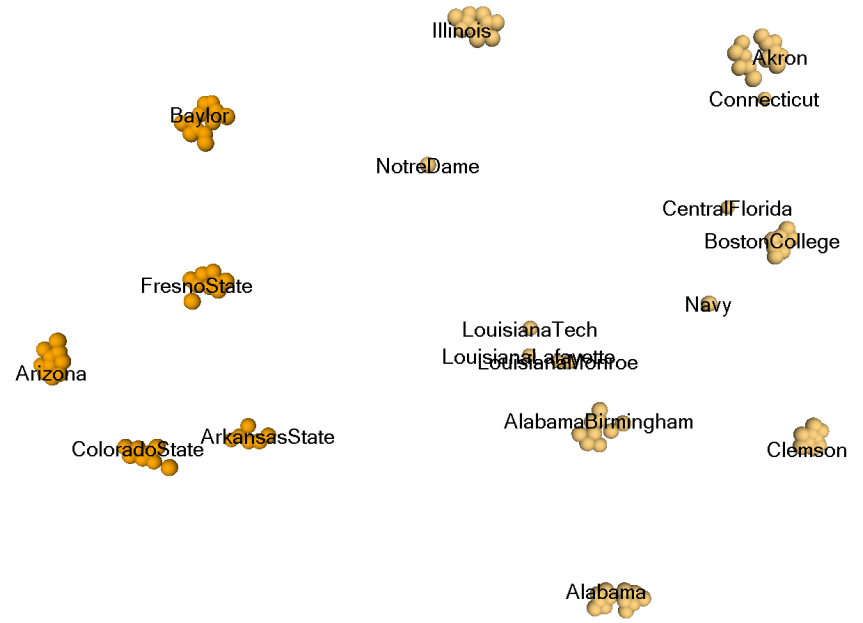


(b) Layout with $Q^{\text{normendv}}(p) = 0.444$, and clustering with $Q^{\text{normendv}}(p) = 0.291$

Figure 3.8: Assignments of the Food Classification graph



(a) Ordering with $Q^{\text{normendv}}(p) = 0.276$, and clustering with $Q^{\text{normendv}}(p) = 0.196$



(b) Layout with $Q^{\text{normendv}}(p) = 0.267$, and clustering with $Q^{\text{normendv}}(p) = 0.196$

Figure 3.9: Assignments of the College Football graph

3.4 Summary

3.4.1 Main Results

- Uniform quality measures are enabled by the unification of graph clusterings, graph orderings, and graph layouts to graph assignments.
- Unbiased quality measures, i.e. quality measures that take the same value for all assignments of all graphs with uniform density, can be derived by normalizing biased quality measures with their value for graphs with uniform density.
- The uniform quality measures and their biases are summarized in Subsection 3.4.2.
- The normalized atedge length, as an unbiased quality measure, is the quotient of the average atedge length and the average atvertex distance, or equivalently the quotient of the total atedge length of the given graph and the total atedge length of a uniform-density graph with the same atvertices and the same number of atedges in the same assignment.
- Quality measures for graphs without vertex weights can be derived as special cases of quality measures for graphs with vertex weights, by assuming unit vertex weights or unit endvertex weights. The resulting quality measures and their biases are summarized in Subsection 3.4.3.
- Empirical observations for several real-world graphs confirm that the grouping of the vertices in clusterings, orderings, and layouts with small normalized atedge length is consistent.

3.4.2 Measures for Graphs

All measures are defined for a graph $G = (\mathcal{V}, \mathcal{E})$, an assignment p, and a distance measure d.

Total atedge length:

$$\mathrm{Q}(\mathrm{p}) \;=\; \sum\nolimits_{\{v_1,v_2\}\in\mathcal{E}} \mathrm{d}(\mathrm{p}(v_1), \mathrm{p}(v_2))$$

- bias towards assignments with small atvertex distances (Theorem 3.1)
- bias towards graphs with small density (Theorem 3.1)

Scaled atedge length:

$$\mathrm{Q}^{\mathrm{scal}}(\mathrm{p}) \;=\; \frac{\mathrm{Q}(\mathrm{p})}{\sum_{\{v_1,v_2\}\in\mathcal{V}^{\langle 2\rangle}} \mathrm{d}(\mathrm{p}(v_1), \mathrm{p}(v_2))}$$

- no bias towards any assignment for graphs with uniform density (Theorem 3.1)
- bias towards graphs with small density (Theorem 3.1)

Normalized atedge length:

$$\mathrm{Q}^{\mathrm{norm}}(\mathrm{p}) \;=\; \mathrm{Q}^{\mathrm{scal}}(\mathrm{p}) \;\Big/\; \frac{|\mathcal{E}|}{|\mathcal{V}^{\langle 2\rangle}|}$$

- no bias towards any assignment for graphs with uniform density (Theorem 3.1)
- no bias towards graphs with particular vertex weights or density (Theorem 3.1)

3.4.3 Measures for Graphs without Vertex Weights

All measures are defined for a graph without vertex weights $G = (V, \mathcal{E})$, an assignment p, and a distance measure d. The quality measures are derived by transforming G into a graph with vertex weights using either unit vertex weights or unit endvertex weights, and applying the quality measures for graphs with vertex weights to the transformed graph.

Total atedge length:

$$\mathrm{Q}(\mathrm{p}) \;=\; \sum\nolimits_{\{v_1,v_2\}\in\mathcal{E}} \mathrm{d}(\mathrm{p}(v_1),\mathrm{p}(v_2))$$

- bias towards assignments with small vertex distances and towards graphs with small density$^{\mathrm{vert}}$ (Theorem 3.1$^{\mathrm{vert}}$)
- bias towards assignments with small endvertex distances and towards graphs with small density$^{\mathrm{endv}}$ (Theorem 3.1$^{\mathrm{endv}}$)

Scaled$^{\mathrm{vert}}$ atedge length:

$$\mathrm{Q}^{\mathrm{scalvert}}(\mathrm{p}) \;=\; \frac{\mathrm{Q}(\mathrm{p})}{\sum_{\{v_1,v_2\}\in V^{(2)}} \mathrm{d}(\mathrm{p}(v_1),\mathrm{p}(v_2))}$$

- no bias towards any assignment for graphs with uniform density$^{\mathrm{vert}}$ (Theorem 3.1$^{\mathrm{vert}}$)
- bias tow. high-degree vertices on central positions for graphs with uniform density$^{\mathrm{endv}}$ (Thm. 3.2)
- bias towards graphs with small density$^{\mathrm{vert}}$ (Theorem 3.1$^{\mathrm{vert}}$)

Normalized$^{\mathrm{vert}}$ atedge length:

$$\mathrm{Q}^{\mathrm{normvert}}(\mathrm{p}) \;=\; \mathrm{Q}^{\mathrm{scalvert}}(\mathrm{p}) \;\Big/\; \frac{|\mathcal{E}|}{|V^{(2)}|}$$

- no bias towards any assignment for graphs with uniform density$^{\mathrm{vert}}$ (Theorem 3.1$^{\mathrm{vert}}$)
- bias tow. high-degree vertices on central positions for graphs with uniform density$^{\mathrm{endv}}$ (Thm. 3.2)
- no bias towards graphs with a particular number of vertices or edges (Theorem 3.1$^{\mathrm{vert}}$)

Scaled$^{\mathrm{endv}}$ atedge length:

$$\mathrm{Q}^{\mathrm{scalendv}}(\mathrm{p}) \;=\; \frac{\mathrm{Q}(\mathrm{p})}{\sum_{\{v_1,v_2\}\in(V,\deg)^{\langle 2\rangle}} \mathrm{d}(\mathrm{p}(v_1),\mathrm{p}(v_2))}$$

- no bias towards any assignment for graphs with uniform density$^{\mathrm{endv}}$ (Theorem 3.1$^{\mathrm{endv}}$)
- no bias towards any assignment for graphs with uniform density$^{\mathrm{vert}}$ (Theorem 3.3)
- bias towards graphs with small density$^{\mathrm{endv}}$ (Theorem 3.1$^{\mathrm{endv}}$)

Normalized$^{\mathrm{endv}}$ atedge length:

$$\mathrm{Q}^{\mathrm{normendv}}(\mathrm{p}) \;=\; \mathrm{Q}^{\mathrm{scalendv}}(\mathrm{p}) \;\Big/\; \frac{|\mathcal{E}|}{|(V,\deg)^{\langle 2\rangle}|}$$

- no bias towards any assignment for graphs with uniform density$^{\mathrm{endv}}$ (Theorem 3.1$^{\mathrm{endv}}$)
- no bias towards any assignment for graphs with uniform density$^{\mathrm{vert}}$ (Theorem 3.3)
- no bias towards graphs with particular vertex degrees (Theorem 3.1$^{\mathrm{endv}}$)

Chapter 4

Quality Measures for Graph Clusterings I

Graph clusterings are graph assignments with a specific distance measure for vertex positions; thus the assignment quality measures of the previous chapter can be specialized to clustering quality measures by inserting this distance measure. The first section of this chapter shows how the values of the resulting clustering quality measures can be interpreted, and how they are reflected in graph visualizations. The second subsection discusses clustering quality measures that result from alternative choices of the distance measure. The third section surveys related clustering quality measures from the literature, and checks them for biases using the technique of the previous chapter. Finally, example clusterings for real-world graphs are presented.

4.1 Simplified Definitions and Visualization

Graph clusterings are graph assignments with the associated distance measure $\overline{\delta}$, which is 0 if its two parameters are equal and 1 otherwise (see Subsection 2.3.2). The quality measures for graph assignments of the previous chapter can be slightly simplified by inserting this specific distance function. Subsection 4.5.2 summarizes the simplified formulations of the measures and their properties. It provides two formulations of each measure, one in terms of the base graph and one in terms of the cluster graph, which are not only mathematically equivalent but also textually similar. The textual similarity is enabled by the definition of the vertex weights and edge weights in the cluster graph as sum of the respective weights in the base graph (in Subsection 2.3.2).

The remainder of this section informally restates the assignment quality measures of the previous chapter and some of their properties in clustering terminology, shows how their values are represented in graph visualizations, and applies them to simple example clusterings.

4.1.1 The Total Atedge Length (Total Atedge Cut)

Definition When applied to clusterings, the total atedge length is also denoted as *total atedge cut* or, if no confusion with the total atpair cut defined in Chapter 7 is possible, simply as *total cut*.[1] It equals the number of the inter-cluster atedges.

According to Theorem 3.1 the total cut is biased towards coarse-grained clusterings. More precisely, the total cut is proportional to the number of inter-cluster atvertex pairs (i.e. pairs of atvertices where each atvertex belongs to a different cluster) for graphs with uniform density. This means that even fixing the number of clusters, e.g. to 2, does not remove the bias, because the number of inter-cluster atvertex pairs is still greater for two similarly-sized clusters than for a small and a large cluster.

Visualization In box-line visualizations of a graph, the weight of each edge equals the width of its representing line (see Subsection 2.4.2). Thus the total cut equals the total width of the edge lines that cross a cluster boundary line in box-line visualizations of the base graph (assuming that each edge line crosses each cluster boundary line at most once), and the total width of the edge lines between different vertices in box-line visualizations of the cluster graph.

In matrix visualizations, each edge (except loops) is represented by one matrix element above the diagonal and one matrix element below the diagonal, and the weight of the edge equals the color weight in each of these elements (see Subsection 2.4.1). All edges between two different clusters are represented by two so-called *cluster-pair rectangles*, which are fatly bounded rectangles in visualizations of the base graph, and single matrix elements in visualizations of the cluster graph. So the total cut equals the total color weight in the cluster-pair rectangles above the diagonal (or equivalently below the diagonal), in matrix visualizations of both the base graph and the cluster graph.

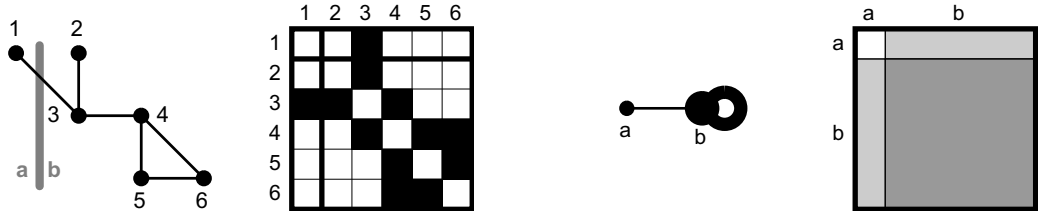
Examples In both clusterings in Figure 4.1, the total cut is 1.

4.1.2 The Scaled Atedge Length (Scaled Atedge Cut) and Normalized Atedge Length (Normalized Atedge Cut)

Definition When applied to clusterings, the scaled atedge length is also denoted as *scaled atedge cut* or simply as *scaled cut*, and equals the ratio of the number of inter-cluster atedges to the number of inter-cluster atvertex pairs, i.e. the inter-cluster density. In contrast to the total cut, the scaled cut is not biased towards any particular clustering for graphs with uniform density (by Theorem 3.1).

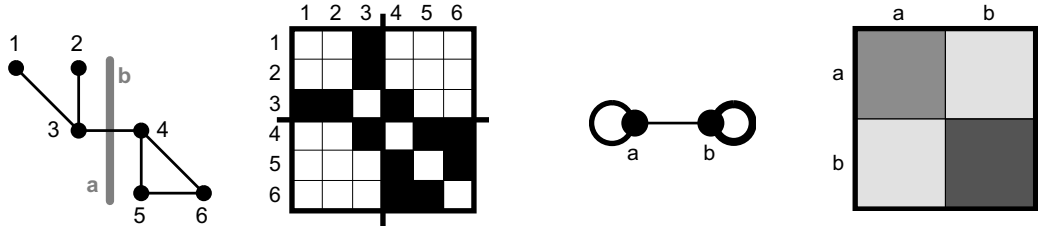
The normalized atedge length is also denoted as *normalized atedge cut* or *normalized cut*, and equals the ratio of the inter-cluster density to the density of the graph. It is additionally not biased towards graphs with a particular density.

[1] This deviates slightly from the standard use of the term cut in graph theory, where it usually denotes the set (not total weight) of inter-cluster edges in clusterings with two clusters.



Base graph, box-line | Base graph, matrix | Cluster graph, box-line | Cluster graph, matrix

Clustering p_1, with $Q(p_1) = 1$ and $Q^{norm}(p_1) = \frac{1}{5}/\frac{6}{15} = 0.5$



Base graph, box-line | Base graph, matrix | Cluster graph, box-line | Cluster graph, matrix

Clustering p_2, with $Q(p_2) = 1$ and $Q^{norm}(p_2) = \frac{1}{9}/\frac{6}{15} \approx 0.28$

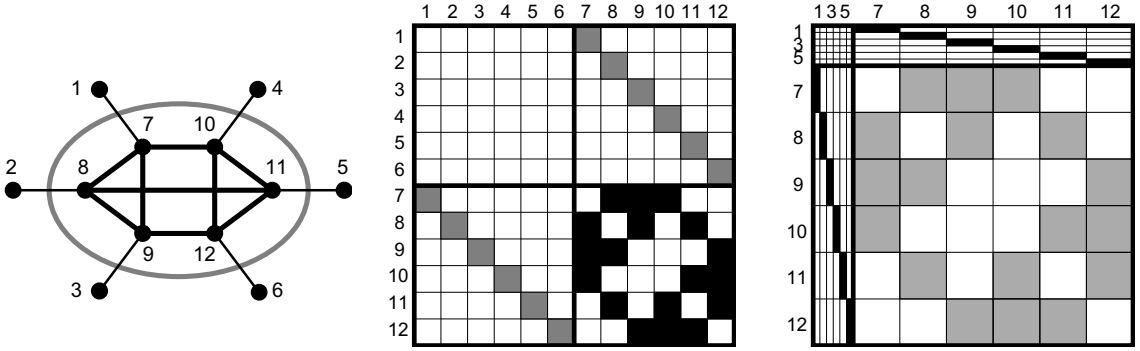
Figure 4.1: Clusterings of a graph with unit vertex weights and unit edge weights

Visualization For both matrix visualizations of the base graph and matrix visualizations of the cluster graph, the total cut equals the total color weight in the cluster-pair rectangles above the diagonal. The number of inter-cluster atvertex pairs equals the total area of the cluster-pair rectangles above the diagonal. So the scaled cut, which is the quotient of these two terms, equals the average color density of the cluster-pair rectangles above the diagonal, or equivalently the average color density of all off-diagonal cluster-pair rectangles.

Examples Both clusterings in Figure 4.1 have the same total cut 1. The normalized cut of both clusterings is smaller than 1 (0.5 and 0.28), so the inter-cluster density is smaller than the density of the graph (and thus the intra-cluster density is larger than the density of the graph). The second clustering has a smaller (i.e. better) normalized cut than the first clustering because of the larger number of inter-cluster atvertex pairs. The smaller normalized cut is reflected by the smaller average color density of the off-diagonal cluster-pair rectangles in the matrix visualizations.

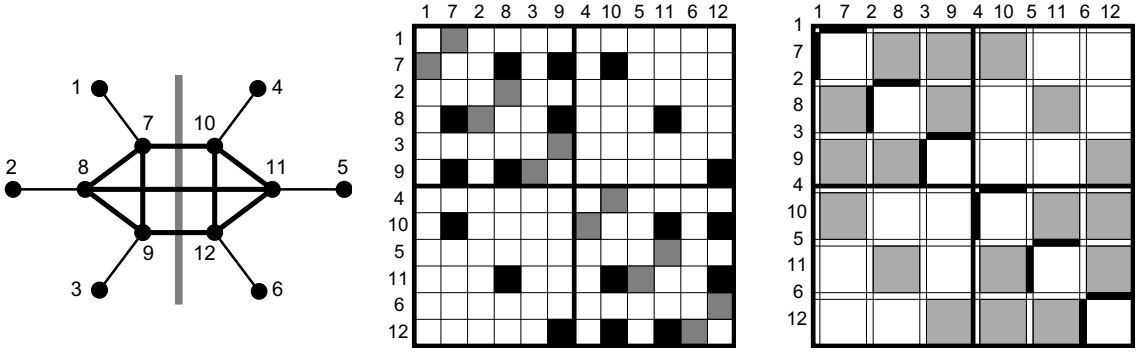
4.1.3 On Graphs without Vertex Weights

As illustrated in Subsection 3.4.3 for general graph assignments, the scaled cut splits into the scaled^{vert} cut and the scaled^{endv} cut, and the normalized cut splits into the normalized^{vert} cut and the normalized^{endv} cut, for graphs without vertex weights. Only matrix^{vert} visualizations reflect the scaled^{vert} and the normalized^{vert} cut, and only matrix^{endv} visualizations reflect the scaled^{endv} and the normalized^{endv} cut.



Box-line$^{\text{vert}}$ Matrix$^{\text{vert}}$ Matrix$^{\text{endv}}$

Clustering p_1, a good clustering with respect to the normalized$^{\text{vert}}$ cut;
$Q(p_1) = 6$, $Q^{\text{normvert}}(p_1) = \frac{6}{36}/\frac{24}{66} \approx 0.46$, $Q^{\text{normendv}}(p_1) = \frac{6}{252}/\frac{24}{1002} \approx 0.99$



Box-line$^{\text{vert}}$ Matrix$^{\text{vert}}$ Matrix$^{\text{endv}}$

Clustering p_2, an optimal clustering with respect to the normalized$^{\text{endv}}$ cut;
$Q(p_2) = 6$, $Q^{\text{normvert}}(p_2) = \frac{6}{36}/\frac{24}{66} \approx 0.46$, $Q^{\text{normendv}}(p_2) = \frac{6}{576}/\frac{24}{1002} \approx 0.43$

Figure 4.2: Clusterings of a graph without vertex weights. The edges between the six central vertices have weight 2, the other edges have weight 1.

The bias of the normalized$^{\text{vert}}$ cut towards assignments with high-degree vertices in large clusters (Theorem 3.2), and the absence of this bias in the normalized$^{\text{endv}}$ cut (Theorem 3.1$^{\text{endv}}$), can be illustrated with clusterings that contain a singleton cluster. Consider the graph in Figure 4.1 as graph without vertex weights. Then the clustering that separates the low-degree vertex 1 from the remaining vertices has the normalized$^{\text{vert}}$ cut $\frac{1}{5}/\frac{6}{15} = 0.5$, and the normalized$^{\text{endv}}$ cut $\frac{1}{11}/\frac{6}{58} \approx 0.88$. The clustering that separates the high-degree vertex 3 from the remaining vertices has the normalized$^{\text{vert}}$ cut $\frac{3}{5}/\frac{6}{15} = 1.5$, and the normalized$^{\text{endv}}$ cut $\frac{3}{27}/\frac{6}{58} \approx 1.07$. So the normalized$^{\text{vert}}$ cut is proportional to the degree of the vertex in the small cluster, while the normalized$^{\text{endv}}$ cut is only weakly dependent from this degree. This weak dependence arises because of the restriction to a singleton cluster, which has no intra-cluster edges.

As a more complex example, Figure 4.2 shows two clusterings of a graph without vertex weights, which are ranked differently by the normalized$^{\text{vert}}$ cut and the normalized$^{\text{endv}}$ cut. The normalized$^{\text{vert}}$ cut is 0.46 and thus smaller than 1 for both clusterings, which means that the inter-cluster density$^{\text{vert}}$ is smaller than the density$^{\text{vert}}$ of the graph. The normalized$^{\text{endv}}$ cut is approximately 1 for the first clustering, which means that the inter-cluster density$^{\text{endv}}$ is about the same as the density$^{\text{endv}}$ of the graph. The reason for this relatively large inter-cluster density$^{\text{endv}}$ (as opposed to the small inter-cluster density$^{\text{vert}}$) is the uneven number of endvertices in the two clusters (as opposed to the even number of vertices), and the resulting relatively small number of inter-cluster endvertex pairs.

4.2 The Impact of the Distance Measure

Graph clusterings are defined as special graph assignments where the distance of two vertices is 0 if their positions (i.e. clusters) are equal, and 1 if their positions are different (see Section 2.3). Assignment quality measures have been specialized to clustering quality measures by inserting this specific distance measure $\overline{\delta}$. However, the constants 0 and 1 are somewhat arbitrary, and particularly the distance measure $\delta = 1 - \overline{\delta}$ is symmetric to $\overline{\delta}$ and therefore equally natural[2]. Two clustering quality measures are called *ranking-equivalent* if they induce the same ranking of the clusterings for any graph (i.e. if their best, second best, etc. clusterings coincide for any graph). The first subsection shows that the $\overline{\delta}$-specialization and the δ-specialization of the normalized atedge length are not ranking-equivalent, and the second subsection introduces a similar assignment quality measure whose specializations are ranking-equivalent.

4.2.1 Impact on the Normalized Atedge Length

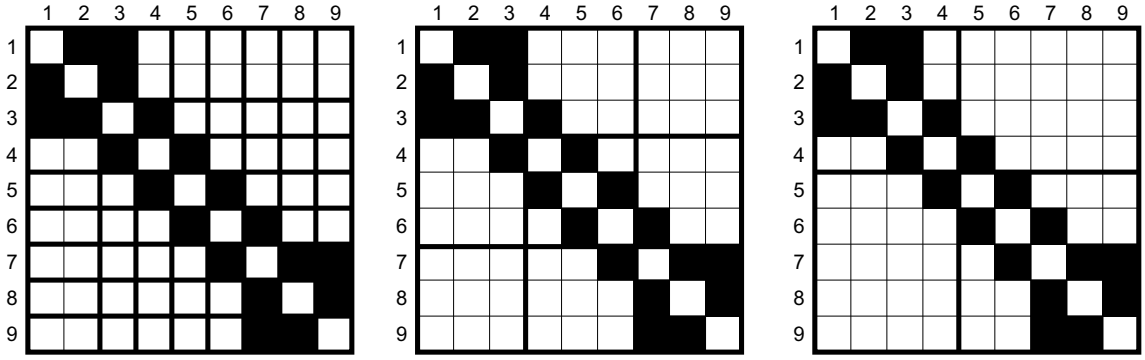
The value of the scaled atedge length and the normalized atedge length clearly does not change if the distance measure is multiplied with any non-zero constant. However, their ranking of clusterings may change significantly if a constant is added to the distance measure. In particular, using $\delta = 1 - \overline{\delta}$ as distance measure, the normalized atedge length of a graph $(\mathcal{V}, \mathcal{E})$ in a clustering p is

$$\frac{\sum_{\{v_1,v_2\}\in\mathcal{E},\ \mathrm{p}(v_1)=\mathrm{p}(v_2)} 1}{\sum_{\{v_1,v_2\}\in\mathcal{V}^{\langle 2\rangle},\ \mathrm{p}(v_1)=\mathrm{p}(v_2)} 1} \Big/ \frac{|\mathcal{E}|}{|\mathcal{V}^{\langle 2\rangle}|} ,$$

i.e. the ratio of the intra-cluster density to the density of the graph (where *higher* values indicate better quality). So the δ-specialization of the normalized atedge length rewards a large intra-cluster density, while the $\overline{\delta}$-specialization penalizes a large inter-cluster density.

[2]Strictly speaking, δ is not a distance measure according to the definition in Section 2.3, because $\delta(p, p) \neq 0$ for all positions p, but this is not critical for the following discussion.

The example in Figure 4.3 shows that the two specializations of the normalized atedge length are not ranking-equivalent: The left clustering has the largest intra-cluster density among all clusterings of the graph, but not the smallest inter-cluster density. The right clustering has the smallest inter-cluster density, but not the largest intra-cluster density.



Clustering p_1, with
$Q^{\mathrm{norm}}_{\delta}(p_1) = \frac{1}{1}/\frac{10}{36} = 3.6$,
$Q^{\mathrm{norm}}_{\overline{\delta}}(p_1) = \frac{9}{35}/\frac{10}{36} \approx 0.93$,
$Q^{\mathrm{shift}}_{\delta}(p_1) = 1 - \frac{10}{36}1 \approx 0.72$,
$Q^{\mathrm{shift}}_{\overline{\delta}}(p_1) = 9 - \frac{10}{36}35 \approx -0.72$

Clustering p_2, with
$Q^{\mathrm{norm}}_{\delta}(p_2) = \frac{8}{9}/\frac{10}{36} = 3.2$,
$Q^{\mathrm{norm}}_{\overline{\delta}}(p_2) = \frac{2}{27}/\frac{10}{36} \approx 0.27$,
$Q^{\mathrm{shift}}_{\delta}(p_2) = 8 - \frac{10}{36}9 = 5.5$,
$Q^{\mathrm{shift}}_{\overline{\delta}}(p_2) = 2 - \frac{10}{36}27 = -5.5$

Clustering p_3, with
$Q^{\mathrm{norm}}_{\delta}(p_3) = \frac{9}{16}/\frac{10}{36} \approx 2.0$,
$Q^{\mathrm{norm}}_{\overline{\delta}}(p_3) = \frac{1}{20}/\frac{10}{36} = 0.18$,
$Q^{\mathrm{shift}}_{\delta}(p_3) = 9 - \frac{10}{36}16 \approx 4.6$,
$Q^{\mathrm{shift}}_{\overline{\delta}}(p_3) = 1 - \frac{10}{36}20 \approx -4.6$

Figure 4.3: Clusterings of a graph with unit vertex weights and unit edge weights

In general, clusterings with small inter-cluster density tend to be coarse-grained, i.e. have few inter-cluster atvertex pairs[3], while clusterings with large intra-cluster density tend to be fined-grained, i.e. have few intra-cluster atvertex pairs. In fact, it can be shown that for each graph (with at least two vertices), at least one clustering with minimal inter-cluster density has two clusters; because for each clustering with at least three clusters, joining the two clusters with the largest inter-density never increases the inter-cluster density of the clustering. Similarly, for each graph (with at least two vertices), at least one clustering with maximal intra-cluster density has only singleton clusters and one two-vertex cluster.

The $\overline{\delta}$-specialization of the normalized atedge length (i.e. the normalized cut) is not only a measure of inter-cluster sparsity, but is also related to intra-cluster density, because fewer inter-cluster atedges necessarily mean more intra-cluster atedges. However, it is primarily a measure of inter-cluster sparsity, and similarly the δ-specialization is primarily a measure of intra-cluster density. A balanced combination of inter-cluster sparsity and intra-cluster density can be measured either with both specializations of the normalized atedge length together, or with a single specialization of an assignment quality measure with ranking-equivalent specializations.

[3]This tendency is not captured by the definition of bias in Subsection 1.1.2, because the inter-cluster density of coarse-grained clusterings does *not* tend to be small; only clusterings with a particularly small inter-cluster density (and also clusterings with a particularly large inter-cluster density!) tend to be coarse-grained.

4.2.2 No Impact on the Shifted Atedge Length

In Section 3.1, the normalized atedge length has been derived by dividing the total atedge length through its value for graphs with uniform density. Alternatively, the biases of the total atedge length may also be removed by subtracting its value for graphs with uniform density. Formally, the *shifted atedge length* of a graph $(\mathcal{V}, \mathcal{E})$ in an assignment p with respect to a distance measure d is

$$\mathrm{Q}^{\mathrm{shift}}(\mathrm{p}) \;:=\; \sum\nolimits_{\{v_1,v_2\}\in\mathcal{E}} \mathrm{d}(\mathrm{p}(v_1),\mathrm{p}(v_2)) \;-\; \frac{|\mathcal{E}|}{|\mathcal{V}^{\langle 2\rangle}|} \sum\nolimits_{\{v_1,v_2\}\in\mathcal{V}^{\langle 2\rangle}} \mathrm{d}(\mathrm{p}(v_1),\mathrm{p}(v_2)) \;.$$

According to Theorem 3.1, the shifted atedge length is 0 for all assignments of all graphs with uniform density, and is thus unbiased.

Adding any constant to the distance measure d does not change the shifted atedge length. Multiplying the distance measure d with any constant only multiplies the shifted atedge length with the same constant. Thus the δ-specialization and the $\overline{\delta}$-specialization of the shifted atedge length are ranking-equivalent.

Of the three clusterings in Figure 4.3, the central clustering optimizes (both the δ-specialization and the $\overline{\delta}$-specialization of) the shifted atedge length. Indeed, the two specializations of the normalized atedge length indicate that this clustering best combines a large intra-cluster density and a small inter-cluster density.

The shifted atedge length is appropriate for graph clusterings, but (in contrast to the normalized atedge length) not for other types of graph assignments, like graph layouts. When a layout is scaled by a factor of k (i.e. all distances in the layout are multiplied by k), then the value of the normalized atedge length does not change, but the value of the shifted atedge length changes by a factor of k. Thus layouts of a graph with optimal shifted atedge length generally have infinite distances.

4.3 Related Work

There is a vast literature on the cluster analysis of vectorial and proximity data, which is surveyed in several books [JD88, KR90, AHS96, Gor99, ELL01] and review articles [JMF99, HBV01, Ber02, GR02, XW05].[4] Recent surveys of graph clustering are less numerous, but the articles Alpert and Kahng [AK95] and Pothen [Pot97] are still good overviews of the main approaches, and Newman [New04b] and Gaertler [Gae05] cover more recent developments.

This section focuses on cut-based quality measures for graph clusterings. It excludes algorithms for finding good or optimal clusterings, which can be found in the referenced literature. It also excludes quality measures that assume vectorial or dissimilarity data, and measures that quantify other properties than intra-cluster density and inter-cluster sparsity.

[4] It is important to distinguish cluster analysis (or unsupervised pattern recognition) from discriminant analysis (or supervised pattern recognition) [DHS00, HTF01]. Discriminant analysis constructs a classifier for estimating the cluster membership of objects, using training data comprised of objects with *known* cluster membership.

The first subsection introduces a uniform template that is used in the following subsections to describe the clustering quality measures. The last subsection discusses other surveys and analyses of quality measures for graph clusterings.

4.3.1 Description Template

All formal definitions in this section are given for a clustering p and a cluster graph $((V, \mathrm{w}), (E, \mathrm{f}))$. The base graph is assumed to have no vertex weights, because most related literature is restricted to such base graphs, and may have loops (i.e. edges from a vertex to itself), because this simplifies some measures and biases without changing the conclusions. To avoid confusion, $\mathrm{w}(v)$ is only used to denote the number of base vertices in the cluster $v \in V$, and $\deg(v)$ is always used to denote the total degree of the base vertices in the cluster v.

The clustering quality measures are described in the following format:

Definition: Formal definition of the measure for clusterings with two cluster v_1 and v_2, i.e. for $V = \{v_1, v_2\}$.

References: References to early or prominent works that use the measure, and the names of the measure in these works. (Some measures have been denoted by various different names, and some names have been attached to various different measures.)

Complexity: References to NP-hardness proofs and approximation algorithms for the problem of finding an optimal two-way clustering of a given graph. If the problem is mentioned in the reference books on NP-completeness by Garey and Johnson [GJ79] or on approximation by Ausiello et al. [ACG+99], then the respective problem numbers are given. The goal are pointers to the literature, not new results.

Bias: Value of the measure for base graphs with uniform density, derived as in the proof of Theorem 3.1. Note that Theorem 3.1 assumes base graphs without loops, while this section assumes base graphs with loops. For a base graph with uniform density$^{\mathrm{vert}}$, the number of atedges between two cluster vertices $v_1, v_2 \in V$ is $\mathrm{w}(v_1)\mathrm{w}(v_2)\frac{|(E,\mathrm{f})|}{\frac{1}{2}|(V,\mathrm{w})|^2}$ if $v_1 \neq v_2$, and $\mathrm{w}(v_1)\mathrm{w}(v_2)\frac{|(E,\mathrm{f})|}{|(V,\mathrm{w})|^2}$ if $v_1 = v_2$. For a base graph with uniform density$^{\mathrm{endv}}$, it is $\deg(v_1)\deg(v_2)\frac{|(E,\mathrm{f})|}{\frac{1}{2}|(V,\deg)|^2}$ if $v_1 \neq v_2$, and $\deg(v_1)\deg(v_2)\frac{|(E,\mathrm{f})|}{|(V,\deg)|^2}$ if $v_1 = v_2$. The density$^{\mathrm{endv}}$ is $\frac{|(E,\mathrm{f})|}{\frac{1}{2}|(V,\deg)|^2}$ and thus equals the inverse total degree $\frac{1}{|(V,\deg)|}$ (because $|(E,\mathrm{f})| = \frac{1}{2}|(V,\deg)|$), but is still referred to as density$^{\mathrm{endv}}$ for uniformity.

Intra-Cluster: Definition of the intra-cluster version of the clustering quality measure, and analysis of its ranking-equivalence to the inter-cluster version given under "Definition". (For the intra-cluster version, large values indicate good clusterings.) See Section 4.2 for the definition of ranking-equivalence, and a detailed discussion of the two versions for the normalized cut.

Multiway: Definition of the measure for clusterings with an arbitrary number of clusters. Some measures for two-way clusterings have more than one generalization to multiway clusterings, and some obvious generalizations have (probably non-obvious) undesired properties.

Related: Measures from the literature with similar properties as the present measure.

The following subsections examine two relationships between clustering quality measures: proportionality and ranking-equivalence. Ranking-equivalence was defined in Section 4.2. Two clustering quality measures Q_1 and Q_2 are called *proportional* if they differ only by a positive constant factor for all clusterings of any graph, i.e. if for all graphs without vertex weights G exists a $k \in \mathbb{R}$ with $k > 0$, such that for all clusterings p of G holds $\mathrm{Q}_1(\mathrm{p}) = k\mathrm{Q}_2(\mathrm{p})$. Proportionality is symmetric and transitive, and implies ranking-equivalence. If a clustering minimizes a quality measure up to a constant factor, then it also minimizes all proportional quality measures up to the same factor. Thus approximability results for one measure generalize to all proportional measures. NP-hardness results clearly generalize to ranking-equivalent measures.

The following properties of the two clusters v_1, v_2 in a two-way clustering are exploited repeatedly in the following analyses:

$$\begin{aligned} 2\mathrm{f}(\{v_1, v_1\}) + \mathrm{f}(\{v_1, v_2\}) &= \deg(v_1) \\ \mathrm{f}(\{v_1, v_1\}) + \mathrm{f}(\{v_1, v_2\}) + \mathrm{f}(\{v_2, v_2\}) &= |(E, \mathrm{f})| \\ \deg(v_1) + \deg(v_2) &= |(V, \deg)| \;=\; 2|(E, \mathrm{f})| \\ \mathrm{w}(v_1) + \mathrm{w}(v_2) &= |(V, \mathrm{w})| \end{aligned}$$

The generalization of these properties to multiway clusterings is straightforward.

4.3.2 Absolute Measures

Total Cut

Definition: $\mathrm{Q}(\mathrm{p}) := \mathrm{f}(\{v_1, v_2\})$

References: Total cut in Section 4.1, cut size in most literature.

Complexity: Minimization in P ([GH61], surveys [CGK+97, JRT00]).
Maximization NP-hard even for unit edge weights ([GJ79, ND16], [ACG+99, ND14]).
Minimization NP-hard with constraint $\mathrm{w}(v_1) = \mathrm{w}(v_2)$, even for unit edge weights (Minimum Bisection [GJ79, ND17], survey [DPS02]).

Bias: $\dfrac{|(E, \mathrm{f})|}{\frac{1}{2}|(V, \mathrm{w})|^2} \cdot \mathrm{w}(v_1)\mathrm{w}(v_2)$ and $\dfrac{|(E, \mathrm{f})|}{\frac{1}{2}|(V, \deg)|^2} \cdot \deg(v_1)\deg(v_2)$
for uniform density$^{\mathrm{vert}}$ and uniform density$^{\mathrm{endv}}$, respectively. Thus

- bias towards clusters of uneven weight and graphs with small density$^{\text{vert}}$,
- bias towards clusters of uneven degree and graphs with small density$^{\text{endv}}$.

Intra-Cluster: $\mathrm{f}(\{v_1, v_1\}) + \mathrm{f}(\{v_2, v_2\}) \;=\; |(E, \mathrm{f})| - \mathrm{Q}(\mathrm{p})$

Thus ranking-equivalent with $-\mathrm{Q}$.

Multiway: $\sum_{\{v_1,v_2\}\in V^{(2)}} \mathrm{f}(\{v_1, v_2\})$

If the number of clusters in unrestricted or fixed, minimization is in P [GH61]; if the number of clusters is an input parameter, minimization is NP-hard [GH88], but approximable within factor $2-\frac{2}{|V|}$ [SV91, SV95]. See also [ACG+99, ND19].

Related: Coverage [BGW03, BGW07]: $\dfrac{\sum_{v\in V} \mathrm{f}(\{v, v\})}{|(E, \mathrm{f})|}$.

The denominator $|(E, \mathrm{f})|$ is constant for a fixed graph. The numerator is the intra-cluster version of the multiway total cut. Biased towards coarse-grained clusterings.

Performance [vD00, Chapter 9], [BGW03, BGW07] for graphs with unit edge weights:

$$\frac{\sum_{v\in V} \mathrm{f}(\{v, v\}) \;+\; \sum_{\{v_1,v_2\}\in V^{(2)}} (\mathrm{w}(v_1)\mathrm{w}(v_2) - \mathrm{f}(\{v_1, v_2\}))}{\frac{1}{2}|(V, \mathrm{w})|^2}$$

The denominator is constant for a fixed graph. The numerator rewards not only intra-cluster edges, but also inter-cluster nonedges. Biased towards fine-grained clusterings for sparse graphs (to maximize inter-cluster nonedges), and coarse-grained clusterings for dense graphs (to maximize intra-cluster edges).

4.3.3 Scaled$^{\text{vert}}$ Measures

Minscaled$^{\text{vert}}$ Cut

Definition: $\mathrm{Q}^{\text{minscalvert}}(\mathrm{p}) \;:=\; \dfrac{\mathrm{f}(\{v_1, v_2\})}{\min(\mathrm{w}(v_1), \mathrm{w}(v_2))}$

References: Edge expansion in [LR88, LR99, KVV00, KVV04], quotient of the cut in [ACG+99, ND26].
Minimum over all clusterings of a graph: flux in [LR88, LR99], isoperimetric number in [Moh89][5], sparsest cut in [ARV04][6].

[5]The Cheeger constant [Che70] or isoperimetric number is originally defined for Riemannian manifolds, as minimum ratio between the boundary area and the volume over all regions (with at most half of the total volume) in the manifold. For graphs, the boundary area of v_1 is often defined as $\mathrm{f}(\{v_1, v_2\})$, and the volume of v_1 is defined either as $\mathrm{w}(v_1)$ or as $\deg(v_1)$. In the earlier case, the Cheeger constant or isoperimetric number equals the minimal minscaled$^{\text{vert}}$ cut; in the latter case, it equals the minimal minscaled$^{\text{endv}}$ cut.

[6]The term sparsest cut is used for minima of the scaled$^{\text{vert}}$ cut and for minima of the minscaled$^{\text{vert}}$ cut.

Complexity: Minimization NP-hard even for graphs with unit edge weights [Kai04]. Approximable within $O(\log|V|)$ [LR88, LR99], recently improved to $O\left(\sqrt{\log|V|}\right)$ [ARV04]. See also [ACG+99, ND26].

Bias: $\frac{|(E,\mathrm{f})|}{\frac{1}{2}|(V,\mathrm{w})|^2} \cdot \max(\mathrm{w}(v_1),\mathrm{w}(v_2))$ for uniform density$^{\mathrm{vert}}$. Thus

- bias towards clusters with equal weight,
- bias towards graphs with small density$^{\mathrm{vert}}$.

Scaled$^{\mathrm{vert}}$ Cut

Definition: $\mathrm{Q}^{\mathrm{scalvert}}(\mathrm{p}) := \frac{\mathrm{f}(\{v_1,v_2\})}{\mathrm{w}(v_1)\mathrm{w}(v_2)}$

References: Scaled$^{\mathrm{vert}}$ cut in Section 4.1, density of the cut in [MS90], ratio of the cut[7] in [WC89, WC91], mean cut in [WS01]. Minimum over all clusterings of a graph: sparsest cut in [MS90, LR88, LR99][8], ratio cut in [WC89, WC91].

Complexity: Minimization NP-hard for graphs with arbitrary edge weights [MS90]; apparently no published NP-hardness proof for graphs with unit edge weights. Approximation see minscaled$^{\mathrm{vert}}$ cut. See also [ACG+99, ND23].

Bias: $\frac{|(E,\mathrm{f})|}{\frac{1}{2}|(V,\mathrm{w})|^2}$ for uniform density$^{\mathrm{vert}}$. Thus

- no bias towards any particular clustering,
- bias towards graphs with small density$^{\mathrm{vert}}$.

Intra-Cluster: There are two natural intra-cluster versions:

- $\frac{\mathrm{f}(\{v_1,v_1\})+\mathrm{f}(\{v_2,v_2\})}{\frac{1}{2}\mathrm{w}(v_1)^2+\frac{1}{2}\mathrm{w}(v_2)^2} = \frac{|(E,\mathrm{f})|-\mathrm{w}(v_1)\mathrm{w}(v_2)\mathrm{Q}^{\mathrm{scalvert}}(\mathrm{p})}{\frac{1}{2}\mathrm{w}(v_1)^2+\frac{1}{2}\mathrm{w}(v_2)^2}$
- $\frac{\mathrm{f}(\{v_1,v_1\})}{\frac{1}{2}\mathrm{w}(v_1)^2}+\frac{\mathrm{f}(\{v_2,v_2\})}{\frac{1}{2}\mathrm{w}(v_2)^2}$

The first variant is ranking-equivalent to $-\mathrm{Q}^{\mathrm{scalvert}}$ only for fixed cluster sizes $\mathrm{w}(v_1)$ and $\mathrm{w}(v_2)$, the second variant not even for fixed cluster sizes.

Multiway: There are three natural generalizations to multiway clusterings:

- $\frac{\sum_{\{v_1,v_2\}\in V^{(2)}}\mathrm{f}(\{v_1,v_2\})}{\sum_{\{v_1,v_2\}\in V^{(2)}}\mathrm{w}(v_1)\mathrm{w}(v_2)}$

[7] The term cut ratio has been used mostly for the scaled$^{\mathrm{vert}}$ cut, but also for the scaled$^{\mathrm{vert}}$ cut sum, and for the ratio $\frac{\mathrm{f}_1(\{v_1,v_2\})}{\mathrm{f}_2(\{v_1,v_2\})}$ of the cuts with respect to two edge weight functions f_1 and f_2 in [WS03].

[8] See footnote[6].

- $\frac{1}{|V|} \sum_{v_1 \in V} \frac{\sum_{v_2 \in V \setminus \{v_1\}} \mathrm{f}(\{v_1, v_2\})}{\sum_{v_2 \in V \setminus \{v_1\}} \mathrm{w}(v_1)\mathrm{w}(v_2)}$
- $\frac{1}{|V^{(2)}|} \sum_{\{v_1,v_2\} \in V^{(2)}} \frac{\mathrm{f}(\{v_1, v_2\})}{\mathrm{w}(v_1)\mathrm{w}(v_2)}$

The definition of the scaled$^{\mathrm{vert}}$ cut in Section 4.1 corresponds to the first variant, which was also proposed as cluster ratio in [YCL92, YCL95]. All three variants take the value $\frac{|(E,\mathrm{f})|}{\frac{1}{2}|(V,\mathrm{w})|^2}$ for graphs with uniform density$^{\mathrm{vert}}$, and are thus not biased towards any clustering for such graphs. In the first variant, the contribution of each inter-cluster atedge $\{v_1, v_2\}$ is the same, namely $1 \,/\, \sum_{\{v_1,v_2\} \in V^{(2)}} \mathrm{w}(v_1)\mathrm{w}(v_2)$, while in the second and third variant, the contribution decays with the size of v_1 and v_2 (e.g. it is $\frac{1}{|V^{(2)}|} \,/\, \mathrm{w}(v_1)\mathrm{w}(v_2)$ in the third variant). Thus removing one atedge between two small clusters and adding ten atedges between two large clusters decreases the two latter variants, but increases the earlier variant.

Related: The normalized$^{\mathrm{vert}}$ cut defined in Section 4.1 is proportional to the scaled$^{\mathrm{vert}}$ cut, and removes its bias towards graphs with small density$^{\mathrm{vert}}$.

As for other quality measures in this section, further quality measures can be derived by subtracting the intra-cluster version from the corresponding inter-cluster version (or vice versa). For the scaled$^{\mathrm{vert}}$ cut, such a measure was proposed as modularization quality in [MMR$^+$98][9] (originally for directed graphs, but the adaption to undirected graphs is straightforward):

$$\frac{1}{|V|} \sum_{v \in V} \frac{\mathrm{f}(\{v, v\})}{\frac{1}{2}\mathrm{w}(v)^2} \;-\; \frac{1}{|V^{(2)}|} \sum_{\{v_1,v_2\} \in V^{(2)}} \frac{\mathrm{f}(\{v_1, v_2\})}{\mathrm{w}(v_1)\mathrm{w}(v_2)}$$

This measure subtracts the third multiway variant of the scaled$^{\mathrm{vert}}$ cut from its intra-cluster version.

A modification that removes the inconsistent weighting of the atedges (observed above) is proposed in [BH04]:

$$\frac{\sum_{v \in V} \mathrm{f}(\{v, v\})}{\sum_{v \in V} \frac{1}{2}\mathrm{w}(v)^2} \;-\; \frac{\sum_{\{v_1,v_2\} \in V^{(2)}} \mathrm{f}(\{v_1, v_2\})}{\sum_{\{v_1,v_2\} \in V^{(2)}} \mathrm{w}(v_1)\mathrm{w}(v_2)}$$

This measure subtracts the first multiway variant of the scaled$^{\mathrm{vert}}$ cut from its intra-cluster version.

A naive multiway generalization of the scaled$^{\mathrm{vert}}$ cut is proposed in [RS98]:

$$\frac{\frac{1}{2} \sum_{v_1 \in V} \sum_{v_2 \in V \setminus \{v_1\}} \mathrm{f}(\{v_1, v_2\})}{\prod_{v \in V} \mathrm{w}(v)}$$

This measure is strongly biased towards clusterings where each cluster has two vertices, because for such clusterings the denominator takes the huge value $2^{n/2}$, where n is the number of vertices of the base graph.

[9] In later papers of the same authors, the intra-cluster version of the scaled$^{\mathrm{endv}}$ cut sum is denoted as modularization quality (e.g. [MM06]).

Scaled$^{\text{vert}}$ Cut Sum

Definition: $\mathrm{Q}^{\text{scalsumvert}}(\mathrm{p}) := \frac{\mathrm{f}(\{v_1,v_2\})}{\mathrm{w}(v_1)} + \frac{\mathrm{f}(\{v_1,v_2\})}{\mathrm{w}(v_2)}$

References: Ratio cut in [CSZ93, CSZ94], average cut in [SM97, SM00, SS00].

Complexity: See scaled$^{\text{vert}}$ cut.

Bias: $\frac{2|(E,\mathrm{f})|}{|(V,\mathrm{w})|}$ for uniform density$^{\text{vert}}$. Thus

- no bias towards any particular clustering,
- bias towards base graphs with small average degree. (The base graph has $|(V,\mathrm{w})|$ vertices and a total degree of $2|(E,\mathrm{f})|$.)

Intra-Cluster: $\frac{\mathrm{f}(\{v_1,v_1\})}{\mathrm{w}(v_1)} + \frac{\mathrm{f}(\{v_2,v_2\})}{\mathrm{w}(v_2)}$

Called average association in [SM97, SM00], sum of densities in [HK95]. Not even ranking-equivalent to $-\mathrm{Q}^{\text{scalsumvert}}$ for fixed cluster sizes $\mathrm{w}(v_1)$ and $\mathrm{w}(v_2)$. Minimization NP-hard for graphs with arbitrary edge weights [HK95].

Multiway: $\frac{1}{|V|-1}\sum_{v_1\in V}\frac{\sum_{v_2\in V\setminus\{v_1\}}\mathrm{f}(\{v_1,v_2\})}{\mathrm{w}(v_1)}$

Called $|V|$-way ratio cut in [CSZ93, CSZ94][10], without the factor $\frac{1}{|V|-1}$. The expectation is $\frac{2|(E,\mathrm{f})|}{|(V,\mathrm{w})|}$, thus no bias towards a particular clustering. Each inter-cluster atedge $\{v_1,v_2\}$ contributes $\frac{1}{|V|-1}\left(\frac{1}{\mathrm{w}(v_1)}+\frac{1}{\mathrm{w}(v_2)}\right)$, thus atedges between small clusters are penalized more than atedges between large clusters.

Shifted$^{\text{vert}}$ Cut

Definition: $\mathrm{Q}^{\text{shiftvert}}(\mathrm{p}) := \mathrm{f}(\{v_1,v_2\}) - \frac{\mathrm{w}(v_1)\mathrm{w}(v_2)}{\frac{1}{2}|(V,\mathrm{w})|^2}|(E,\mathrm{f})|$

References: Subsection 4.2.2, which also includes a comparison with the normalized$^{\text{vert}}$ cut.

Complexity: No published results.

Bias: 0 for uniform density$^{\text{vert}}$. Thus

- no bias towards any particular clustering,
- no bias towards graphs with a particular number of vertices or atedges.

Intra-Cluster: $\mathrm{f}(\{v_1,v_1\})+\mathrm{f}(\{v_2,v_2\})-\frac{\mathrm{w}(v_1)^2+\mathrm{w}(v_2)^2}{|(V,\mathrm{w})|^2}|(E,\mathrm{f})| = -\mathrm{Q}^{\text{shiftvert}}(\mathrm{p})$ (!)

[10] See footnote[7].

Multiway: $\sum_{\{v_1,v_2\}\in V^{(2)}} \mathrm{f}(\{v_1,v_2\}) \;-\; \sum_{\{v_1,v_2\}\in V^{(2)}} \frac{\mathrm{w}(v_1)\mathrm{w}(v_2)}{\frac{1}{2}|(V,\mathrm{w})|^2}|(E,\mathrm{f})|$

Relationships The scaled$^{\mathrm{vert}}$ cut is proportional to the scaled$^{\mathrm{vert}}$ cut sum (only) for two-way clusterings:

$$\mathrm{Q}^{\mathrm{scalsumvert}}(\mathrm{p}) = \frac{\mathrm{f}(v_1,v_2)}{\mathrm{w}(v_1)} + \frac{\mathrm{f}(v_1,v_2)}{\mathrm{w}(v_2)} = \frac{(\mathrm{w}(v_1)+\mathrm{w}(v_2))\mathrm{f}(v_1,v_2)}{\mathrm{w}(v_1)\mathrm{w}(v_2)} = |(V,\mathrm{w})|\ \mathrm{Q}^{\mathrm{scalvert}}(\mathrm{p})\ .$$

The scaled$^{\mathrm{vert}}$ cut is proportional to the normalized$^{\mathrm{vert}}$ cut (defined in Section 4.1) by definition.

The minscaled$^{\mathrm{vert}}$ cut is closely related to the scaled$^{\mathrm{vert}}$ cut (and thus to the normalized$^{\mathrm{vert}}$ cut and scaled$^{\mathrm{vert}}$ cut sum) for two-way clusterings:

$$\frac{1}{2}|(V,\mathrm{w})|\ \mathrm{Q}^{\mathrm{scalvert}}(\mathrm{p}) \;\le\; \mathrm{Q}^{\mathrm{minscalvert}}(\mathrm{p}) \;\le\; |(V,\mathrm{w})|\ \mathrm{Q}^{\mathrm{scalvert}}(\mathrm{p})$$

because $\mathrm{Q}^{\mathrm{minscalvert}}(\mathrm{p}) = \max(\mathrm{w}(v_1),\mathrm{w}(v_2)) \cdot \mathrm{Q}^{\mathrm{scalvert}}(\mathrm{p})$
and $\frac{1}{2}|(V,\mathrm{w})| \le \max(\mathrm{w}(v_1),\mathrm{w}(v_2)) \le |(V,\mathrm{w})|$.

4.3.4 Scaled$^{\mathrm{endv}}$ Measures

Minscaled$^{\mathrm{endv}}$ Cut

Definition: $\mathrm{Q}^{\mathrm{minscalendv}}(\mathrm{p}) \;:=\; \dfrac{\mathrm{f}(\{v_1,v_2\})}{\min(\deg(v_1),\deg(v_2))}$

References: Conductance in [KVV00, KVV04, BGW03, BGW07].
Minimum over all clusterings of a graph: Cheeger constant [Chu97][11].

Complexity: Minimization NP-hard even for graphs with unit edge weights [SS06].
Approximable within $O(\log|V|)$ [LR88, LR99], recently improved to $O\left(\sqrt{\log|V|}\right)$ [ARV04].

Bias: $\dfrac{|(E,\mathrm{f})|}{\frac{1}{2}|(V,\deg)|^2} \cdot \max(\deg(v_1),\deg(v_2))$ for uniform density$^{\mathrm{endv}}$. Thus

- bias towards clusters with equal degree,
- bias towards graphs with small density$^{\mathrm{endv}}$.

Scaled$^{\mathrm{endv}}$ Cut

Definition: $\mathrm{Q}^{\mathrm{scalendv}}(\mathrm{p}) \;:=\; \dfrac{\mathrm{f}(\{v_1,v_2\})}{\deg(v_1)\deg(v_2)}$

References: Section 4.1.

[11] See footnote[5].

Complexity: Minimization NP-hard for graphs with arbitrary edge weights [SM00] (via scaled$^{\text{endv}}$ cut sum); apparently no published NP-hardness proof for graphs with unit edge weights. Approximation see minscaled$^{\text{endv}}$ cut.

Bias: $\frac{|(E,\mathrm{f})|}{\frac{1}{2}|(V,\deg)|^2}$ for uniform density$^{\text{endv}}$. Thus

- no bias towards any particular clustering,
- bias towards graphs with small density$^{\text{endv}}$.

Intra-Cluster: Similar to the scaled$^{\text{vert}}$ cut.

Multiway: Similar to the scaled$^{\text{vert}}$ cut.

Related: The normalized$^{\text{endv}}$ cut defined in Section 4.1 is proportional to the scaled$^{\text{endv}}$ cut, and removes its bias towards graphs with small density$^{\text{endv}}$.

Scaled$^{\text{endv}}$ Cut Sum

Definition: $\mathrm{Q}^{\text{scalsumendv}}(\mathrm{p}) := \frac{\mathrm{f}(\{v_1,v_2\})}{\deg(v_1)} + \frac{\mathrm{f}(\{v_1,v_2\})}{\deg(v_2)}$

References: Normalized cut in [SM97, SM00].

Complexity: See scaled$^{\text{endv}}$ cut.

Bias: 1 for uniform density$^{\text{endv}}$. Thus

- no bias towards any particular clustering,
- no bias towards graphs with particular degrees.

Intra-Cluster:

$$\begin{aligned}\frac{\mathrm{f}(\{v_1,v_1\})}{\deg(v_1)} + \frac{\mathrm{f}(\{v_2,v_2\})}{\deg(v_2)} &= \frac{\frac{1}{2}(\deg(v_1)-\mathrm{f}(\{v_1,v_2\}))}{\deg(v_1)} + \frac{\frac{1}{2}(\deg(v_2)-\mathrm{f}(\{v_1,v_2\}))}{\deg(v_2)} \\ &= 1-\frac{1}{2}\mathrm{Q}^{\text{scalsumendv}}(\mathrm{p})\end{aligned}$$

Called normalized association in [SM97, SM00]. Ranking-equivalent to $-\mathrm{Q}^{\text{scalsumendv}}$.

Multiway: $\frac{1}{|V|-1}\sum_{v_1\in V}\frac{\sum_{v_2\in V\setminus\{v_1\}}\mathrm{f}(\{v_1,v_2\})}{\deg(v_1)}$

Called multiway normalized cut in [MX03] without the factor $\frac{1}{|V|-1}$, and $|V|$-way normalized cut in [YS03] with a factor $\frac{1}{|V|}$ instead of $\frac{1}{|V|-1}$. The expectation is 1, thus no bias towards any clustering (but there are biases towards few clusters in the variant of [MX03] and many clusters in the variant of [YS03]). Each inter-cluster atedge $\{v_1,v_2\}$ contributes $\frac{1}{|V|-1}\left(\frac{1}{\deg(v_1)}+\frac{1}{\deg(v_2)}\right)$, thus atedges between small clusters are penalized more than atedges between large clusters.

Remark: There is actually a subtle but crucial difference between the scaled$^{\text{endv}}$ cut sum and the normalized cut in [SM97, SM00, YS03]: While the definition of the degree $\deg(v)$ in Section 2.2 counts the atloops of v twice, the other works use a notion of degree $\deg'(v)$ where atloops are counted only once, i.e. $\deg(v) = \deg'(v) + \mathrm{f}(\{v, v\})$. The latter notion is problematic because the degree of a cluster vertex is generally *not* equal to the total degree of its base vertices. The normalized cut approximates the scaled$^{\text{endv}}$ cut sum for fine-grained clusterings, because they have few atloops. Large clusters v tend to have many atloops, thus $\deg'(v)$ tends to be significantly smaller than $\deg(v)$, and the normalized cut tends to be larger than the scaled$^{\text{endv}}$ cut sum for coarse-grained clusterings. Thus the original normalized cut is biased towards fine-grained clusterings. In other words, the scaled$^{\text{endv}}$ cut sum is a new and improved (namely, unbiased) version of the normalized cut obtained by replacing $\deg'(v)$ with $\deg(v)$.

Shifted$^{\text{endv}}$ Cut

Definition: $\mathrm{Q}^{\text{shiftendv}}(\mathrm{p}) := \mathrm{f}(\{v_1, v_2\}) - \dfrac{\deg(v_1)\deg(v_2)}{\frac{1}{2}|(V, \deg)|^2}|(E, \mathrm{f})|$

References: Subsection 4.2.2.

Complexity: Minimization NP-hard even for graphs with unit edge weights [BDG$^+$07] (via Newman's modularity, which is defined below).

Bias: 0 for uniform density$^{\text{endv}}$. Thus

- no bias towards any particular clustering,
- no bias towards graphs with particular degrees.

Intra-Cluster: Similar to the shifted$^{\text{vert}}$ cut.

Multiway: Similar to the shifted$^{\text{vert}}$ cut.

Related: A measure that differs only by the constant factor $\frac{1}{|(E,\mathrm{f})|}$ from the intra-cluster version of the shifted$^{\text{endv}}$ cut was introduced as modularity by Newman [New04a]. (This version of the measure differs slightly from an earlier version in [NG04].)

Recently, Newman [New06] and Reichardt and Bornholdt [RB06] have defined the class of clustering quality measures that subtract the expected total cut (in some null model) from the actual total cut. The shifted cut (with the shifted$^{\text{vert}}$ and the shifted$^{\text{endv}}$ cut as special cases) instantiates this class for graphs of uniform density as null model.

Relationships The scaled$^{\text{endv}}$ cut is proportional to the scaled$^{\text{endv}}$ cut sum (only) for two-way clusterings:

$$\begin{aligned} \mathrm{Q}^{\text{scalsumendv}}(\mathrm{p}) &= \frac{\mathrm{f}(v_1, v_2)}{\deg(v_1)} + \frac{\mathrm{f}(v_1, v_2)}{\deg(v_2)} \\ &= \frac{(\deg(v_1)+\deg(v_2))\mathrm{f}(v_1, v_2)}{\deg(v_1)\deg(v_2)} \;=\; |(V, \deg)|\; \mathrm{Q}^{\text{scalendv}}(\mathrm{p}) \;. \end{aligned}$$

The scaled$^{\text{endv}}$ cut is proportional to the normalized$^{\text{endv}}$ cut (defined in Section 4.1) by definition.

The minscaled$^{\text{endv}}$ cut is closely related to the scaled$^{\text{endv}}$ cut (and thus the normalized$^{\text{endv}}$ cut and scaled$^{\text{endv}}$ cut sum) for two-way clusterings:

$$\frac{1}{2}|(V, \deg)|\; \mathrm{Q}^{\text{scalendv}}(\mathrm{p}) \;\le\; \mathrm{Q}^{\text{minscalendv}}(\mathrm{p}) \;\le\; |(V, \deg)|\; \mathrm{Q}^{\text{scalendv}}(\mathrm{p})$$

because $\mathrm{Q}^{\text{minscalendv}}(\mathrm{p}) = \max(\deg(v_1), \deg(v_2)) \cdot \mathrm{Q}^{\text{scalendv}}(\mathrm{p})$
and $\frac{1}{2}|(V, \deg)| \le \max(\deg(v_1), \deg(v_2)) \le |(V, \deg)|$.

4.3.5 Scaled$^{\text{edge}}$ Measures

Only the properties of the scaled$^{\text{edge}}$ cut sum are detailed in this subsection. The minscaled$^{\text{edge}}$ cut, the scaled$^{\text{edge}}$ cut, and the shifted$^{\text{edge}}$ cut could be defined and analyzed similarly to the minscaled$^{\text{vert}}$ cut, the scaled$^{\text{vert}}$ cut, and the shifted$^{\text{vert}}$ cut. They have not been proposed in the literature.

Scaled$^{\text{edge}}$ Cut Sum

Definition: $\mathrm{Q}^{\text{scalsumedge}}(\mathrm{p}) \;:=\; \dfrac{\mathrm{f}(\{v_1, v_2\})}{\mathrm{f}(\{v_1, v_1\})} + \dfrac{\mathrm{f}(\{v_1, v_2\})}{\mathrm{f}(\{v_2, v_2\})}$

References: Min-max cut in [DHZ$^+$01].

Complexity: No published results.

Bias: $\dfrac{\mathrm{w}(v_2)}{\frac{1}{2}\mathrm{w}(v_1)} + \dfrac{\mathrm{w}(v_1)}{\frac{1}{2}\mathrm{w}(v_2)}$ and $\dfrac{\deg(v_2)}{\frac{1}{2}\deg(v_1)} + \dfrac{\deg(v_1)}{\frac{1}{2}\deg(v_2)}$
for uniform density$^{\text{vert}}$ and uniform density$^{\text{endv}}$, respectively. Thus

- bias towards clusters with equal weight and equal degree[12],
- no bias towards graphs with particular degrees.

Intra-Cluster: $\dfrac{\mathrm{f}(\{v_1, v_1\})}{\mathrm{f}(\{v_1, v_1\})} + \dfrac{\mathrm{f}(\{v_2, v_2\})}{\mathrm{f}(\{v_2, v_2\})} \;=\; 2$

Thus trivial, and not ranking-equivalent to $-\mathrm{Q}^{\text{scalsumedge}}$.

[12]This tendency to produce "balanced" cuts was also recognized by the inventors of the measure [DHZ$^+$01], and considered as an advantage.

Multiway: $\sum_{v_1 \in V} \frac{\sum_{v_2 \in V \setminus \{v_1\}} \mathrm{f}(\{v_1, v_2\})}{\mathrm{f}(\{v_1, v_1\})}$

Also introduced in [DHZ+01].

4.3.6 Validation of Clustering Quality Measures

Surveys The literature lacks comprehensive and up-to-date surveys of quality measures for graph clusterings. Even the recent introduction to density-based graph clustering by Gaertler [Gae05] focuses only on biased measures, namely on coverage, conductance, and performance. A survey by Boutin and Hascoët [BH04] misses some of the best-known measures (e.g. the scaled$^{\text{vert}}$ cut and the scaled$^{\text{endv}}$ cut sum), and does not provide a systematic evaluation of the listed measures (although a few improvements are suggested).

Theoretical Validation Puzicha et al. [PHB00] derive quality measures for clusterings of dissimilarity data from axioms, similar to the derivation of assignment quality measures from the requirement of absence of bias in Section 3.1 of this work. Their resulting six elementary measures are very similar to the multiway inter-cluster and intra-cluster versions of the scaled$^{\text{vert}}$ cut and the scaled$^{\text{vert}}$ cut sum. They did not find scaled$^{\text{endv}}$ and scaled$^{\text{edge}}$ measures because of their axiom of shift invariance. This axiom requires that adding a constant k to all dissimilarities increases the value of the quality measure by nk, where n is the number of vertices of the base graph. Shift invariance is indeed crucial if the edge weights are only interval-scaled, but unnecessary if they are ratio-scaled (as in the present work). Scaled$^{\text{endv}}$ and scaled$^{\text{edge}}$ clustering quality measures are generally not shift invariant, because an increase of all edge weights affects both their numerator and their denominator.

Kleinberg proposes three properties of *clustering functions*, which take a set of objects with pairwise distances, and return a partition of the set of objects [Kle03]:

- *Richness*: The range of the clustering function contains all partitions of the set of objects.
- *Scale-Invariance*: Multiplying all distances with a positive constant factor does not change the resulting partition.
- *Consistency*: Increasing inter-cluster distances and decreasing intra-cluster distances does not change the resulting partition.

In other words, Scale-Invariance requires that changes of absolute distances are irrelevant as long as the ratios of the distances are unchanged, and Consistency requires that changes of the ratios are irrelevant as long as the absolute distances still satisfy certain constraints. It is thus not really surprising that there is no clustering function with all three properties (for the nontrivial case of at least two objects) [Kle03, Theorem 2.1], but it clarifies inherent trade-offs in the choice of clustering functions. The absence of bias, as the central requirement for clustering quality measures in this work, implies Scale-Invariance (at least for graphs with uniform density), but not Consistency.

Some proposals of new clustering quality measures also provide some theoretical justification. Wei and Cheng [WC89, WC91] show that all graph clusterings with two clusters have the same expected scaled$^{\text{vert}}$ cut, in a simple random graph model where any two different vertices are connected with the same probability. Ding et al. [DHZ+01] use the same random graph model to demonstrate that their scaled$^{\text{edge}}$ cut sum is biased towards balanced clusterings, while the scaled$^{\text{vert}}$ cut sum and the scaled$^{\text{endv}}$ cut sum aren't. Newman [New04a] introduced his modularity measure (discussed above as variant of the shifted$^{\text{endv}}$ cut) as the fraction of intra-cluster edges, minus the *expected* fraction of intra-cluster edges, originally without a precise definition of the random graph model used for computing the expectation.

More recently, Newman [New06] and Reichardt and Bornholdt [RB06] have observed that the *difference* of the actual total cut and the expected total cut (in some null model) defines a general class of clustering quality measures, and derive the shifted$^{\text{vert}}$ cut and the shifted$^{\text{endv}}$ cut as instantiations of this class for two specific null models. This parallels the (independent) observation in Section 3.1 of this work, that unbiased assignment quality measure can be derived as *quotient* of the total atedge length and the expected total atedge length, and the derivation of the normalized$^{\text{vert}}$ and normalized$^{\text{endv}}$ atedge length. The consequences of using differences instead of quotients were already discussed in Subsection 4.2.2. As further distinguishing points, Section 3.1 covers graph assignments, not only graph clusterings, and graphs with vertex weights, not only graphs without vertex weights.

Empirical Validation The results of empirical studies of graph clustering do not only depend on the optimized quality measures, but also on the optimization heuristics (because the computation of global optima is intractable in practice), and on the clustered data. It is thus difficult to design a study that allows general conclusions about clustering quality measures, and hardly any such study has been published.

Ding et al. [DHZ+01] empirically compare their scaled$^{\text{edge}}$ cut sum with the scaled$^{\text{vert}}$ cut sum and the scaled$^{\text{endv}}$ cut sum. They cluster pairwise similarities of newsgroup articles with spectral optimization heuristics. The scaled$^{\text{edge}}$ cut sum performs better than the scaled$^{\text{endv}}$ cut sum, which is unsurprising given that the scaled$^{\text{edge}}$ cut sum is biased towards clusters of equal size, and that the correct clusters have equal or similar size (two clusters of 200 documents in a first experiment, one of 200 and one of 300 articles in a second experiment). Both the scaled$^{\text{edge}}$ cut sum and the scaled$^{\text{endv}}$ cut sum perform clearly better than the scaled$^{\text{vert}}$ cut sum.

An extensive empirical study of quality measures for document clustering has been performed by Zhao and Karypis [ZK04]. Unfortunately, four of the seven studied measures are only applicable to vectorial data, because they rely on cluster centroids. The other three measures are the intra-cluster versions of the scaled$^{\text{vert}}$ cut sum (called $\mathcal{I}_1$ in [ZK04]), the scaled$^{\text{edge}}$ cut sum (called $\mathcal{G}_1$), and the scaled$^{\text{endv}}$ cut sum (called $\mathcal{G}_2$). However, the study does not even allow conclusions about the relative performance of these three measures, because they are applied to different graphs (pairwise similarities between documents for the two earlier measures, relations between terms and documents for the latter measure).

4.4 Examples

This section serves two purposes. First, it illustrates the effects of biases (and their absence) in clustering quality measures for real-world graphs. Second, it provides empirical evidence for the internal and external validity of the normalized$^{\text{vert}}$ cut and the normalized$^{\text{endv}}$ cut as unbiased clustering quality measures.

This section does *not* provide an empirical comparison with existing clustering quality measures. Because of the large number of proposed quality measures and their sometimes subtle differences (see Section 4.3), this would require extensive experiments that are beyond the scope of this work. The normalized cut is not claimed to be more valid than the best existing quality measures; its introduction is motivated by its uniformity with quality measures for graph orderings and graph layouts, and by the interpretability and visualizability of its values.

Method All graphs in this section are described in Appendix A.1. For each graph, a clustering with optimal total cut (except for the trivial clustering with one cluster), a clustering with small normalized$^{\text{vert}}$ cut, and a clustering with small normalized$^{\text{endv}}$ cut are presented. All these clusterings have exactly two clusters, for reasons explained in Subsection 4.2.1. The clusterings with small normalized$^{\text{vert}}$ cut and with small normalized$^{\text{endv}}$ are shown as saturation (or darkness, in gray-scale printouts) of the vertex representations in a matrix$^{\text{vert}}$ and a matrix$^{\text{endv}}$ visualization, respectively. They were computed with a multi-scale heuristic (see e.g. [KK98] for the basic ideas), which – as usual in the practical cluster analysis of nontrivial graphs – does not guarantee that the computed clusterings are optimal; accordingly, their optimality is not assumed in the following.

Validation methods for graph assignments and assignment quality measures were introduced in Section 1.3. Internal validity of a clustering quality measure requires that in clusterings with small measurement values, densely connected vertices belong to the same cluster, and sparsely connected vertices belong to different clusters. This is checked using the matrix visualizations, where the density is represented by the color density of the matrix elements. External validity of a clustering quality measure requires that clusterings with small measurement values are externally valid, i.e. resemble known authoritative clusterings. This is checked by comparing the computed clusterings with the authoritative groupings described in Appendix A.1.

Results The clusterings with minimal total cut, with small normalized$^{\text{vert}}$ cut, and with small normalized$^{\text{endv}}$ cut are shown in Figures 4.4 to 4.9 on pages 64 to 69.

The clusterings with minimal total cut are trivial for all graphs except Southern Women; they only separate a vertex with minimal degree (for Southern Women, three vertices) from the remaining vertices. This bias towards coarse-grained clusterings was predicted by Theorem 3.1. The remainder of this paragraph focuses on the clusterings with small normalized$^{\text{vert}}$ cut and small normalized$^{\text{endv}}$ cut.

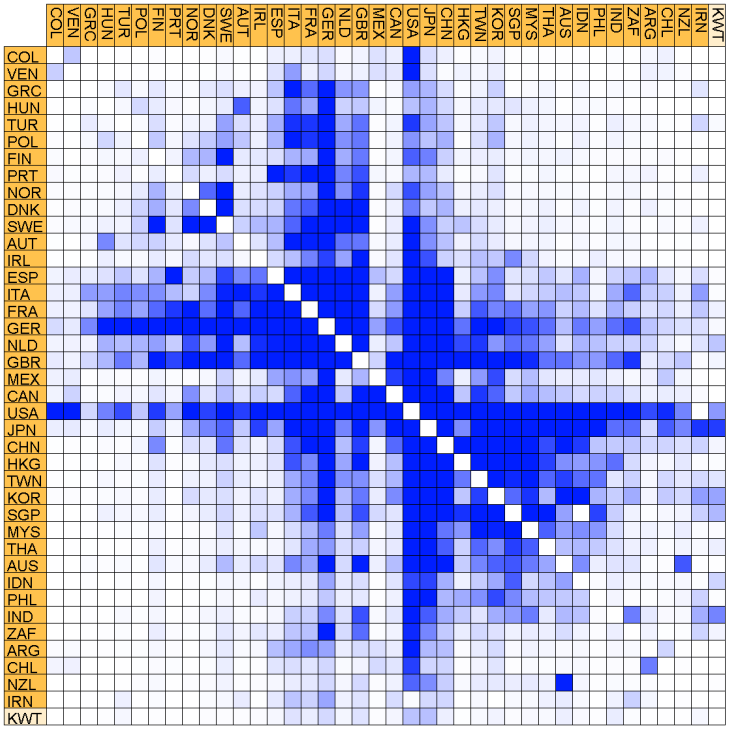
Concerning internal validity, the clusterings with small normalized$^{\text{endv}}$ cut indeed assign most densely$^{\text{endv}}$ connected vertices to the same cluster. This can be seen

in the matrix[endv] visualizations, where most matrix elements with high color density are intra-cluster elements. However, some clusterings fail to assign sparsely[endv] connected vertices to different clusters. Particularly for the College Football graph, the authoritative clustering with 12 clusters corresponding to the conferences shows a similar inter-cluster sparsity[endv] as the computed clustering, but a much greater intra-cluster density[endv]. This focus of the normalized cut on inter-cluster sparsity (as opposed to intra-cluster density) was already observed and explained in Subsection 4.2.1. The observations for the clusterings with small normalized[vert] cut are similar.

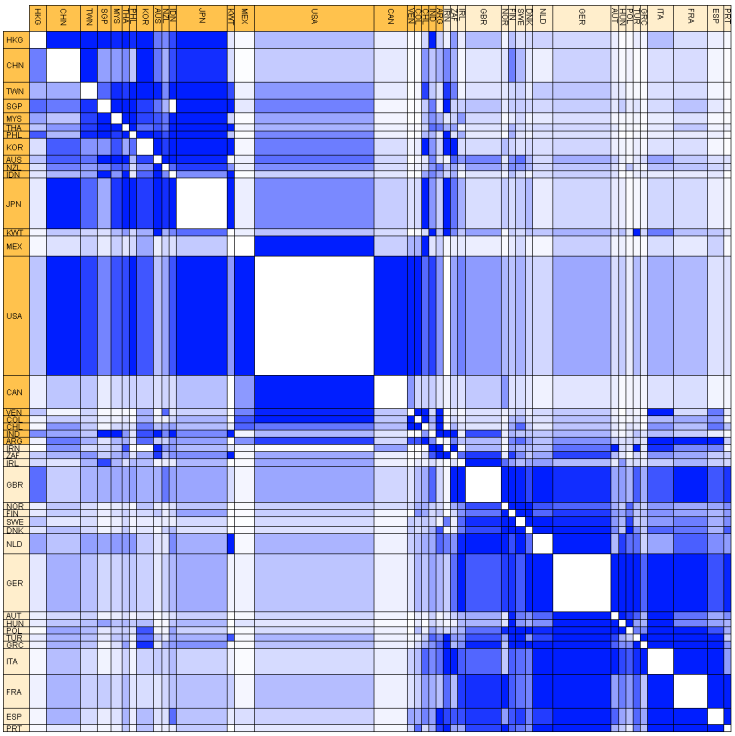
Concerning external validity, the observations for the clusterings with small normalized[endv] cut are as follows. The clustering of World Trade separates East Asia, Australia, and America from Europe and West Asia. This conforms to the authoritative clustering, although an additional separation of East Asia from America would probably be even better. The clustering of Southern Women is identical to the result of Freeman's consensus analysis [Fre03] (for the women; no authoritative clustering is available for the events). The clustering of Karate Club conforms to the authoritative clustering. For Morse Code Confusion, no clear clustering can be derived from the description of the authoritative grouping in Appendix A.1. The clustering of Food Classification conforms to the authoritative clustering by separating fruits and vegetables from other foods. However, an additional separation e.g. of dairies and meat (including seafood) would be desirable. Similarly, the clustering of College Football successfully groups the teams of each conference, but fails to separate teams from different conferences.

As to the clusterings with small normalized[vert] cut, the result for World Trade is trivial. The clustering of Southern Women differs from the result of Freeman's consensus analysis [Fre03]. However, it conforms to the results of some studies cited by Freeman, which assign Olivia and Flora to a separate group or to no group at all. This is a plausible alternative to the consensus clustering because Olivia and Flora attend only to few (namely, two) events. The clusterings of Karate Club, Food Classification, and College Football are identical to the clustering with small normalized[endv] cut. This is unsurprising for the latter two graphs where the vertex degrees are fairly uniform.

In summary, the examined clusterings with small normalized[endv] cut conform well to the authoritative groupings; they differ from the authoritative groupings only in their limitation to two clusters, which was predicted and discussed in Subsection 4.2.1. The clusterings with small normalized[vert] cut differ from the clustering with small normalized[endv] cut for World Trade and Southern Women, where their external validity is worse. The clusterings with minimal total cut are mostly trivial, as predicted by Theorem 3.1.

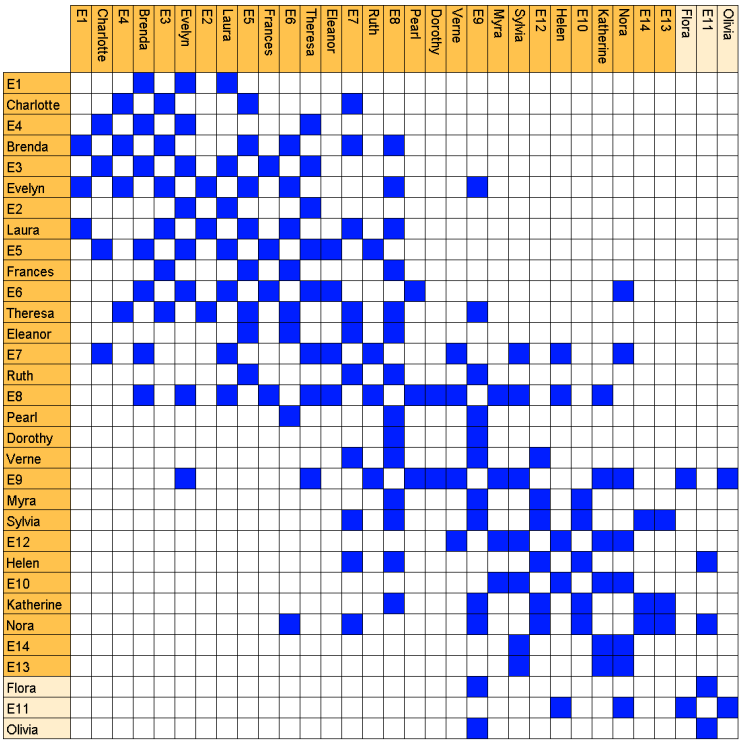


(a) Clustering with minimal total cut $18.6 \cdot 10^9$ and small normalized$^{\text{vert}}$ cut 0.089

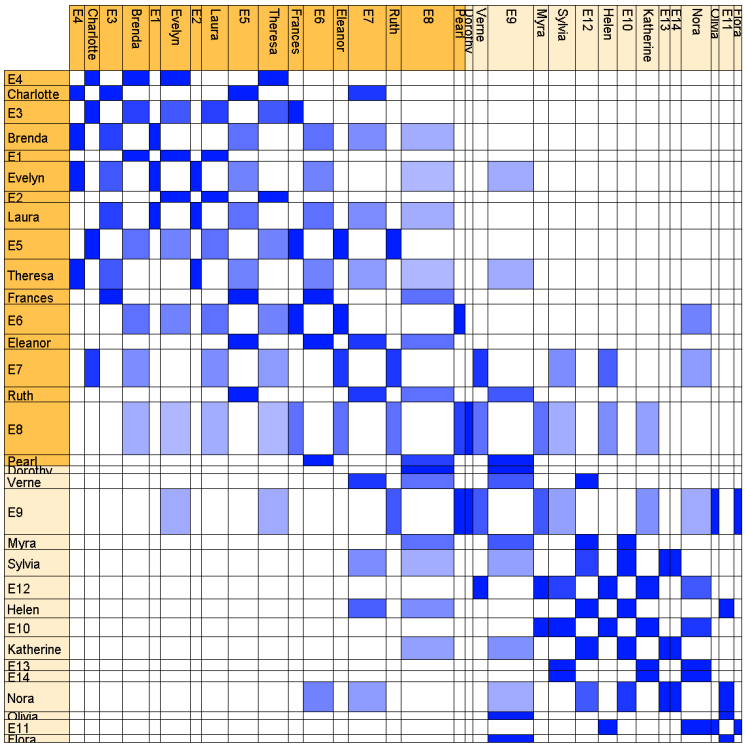


(b) Clustering with small normalized$^{\text{endv}}$ cut 0.425

Figure 4.4: Clusterings of the World Trade graph

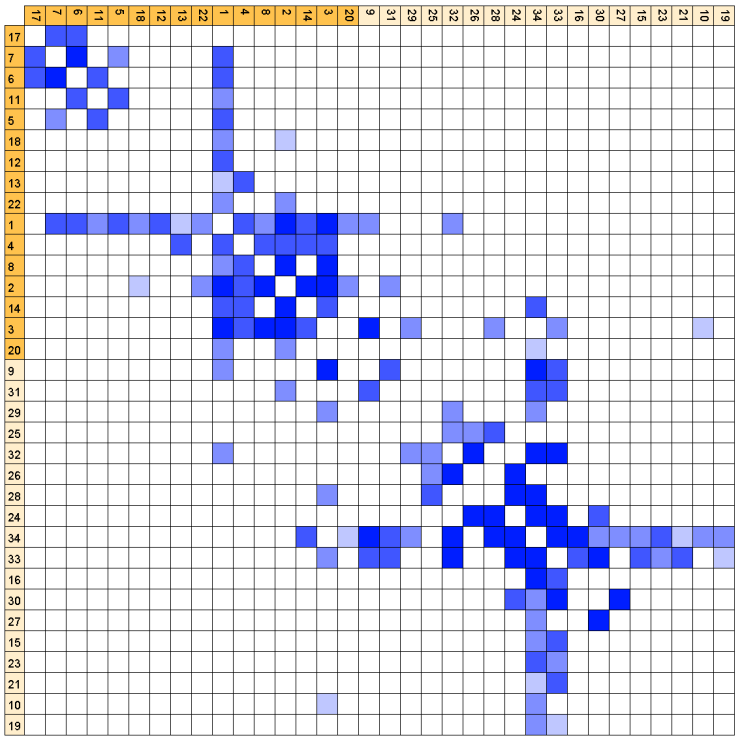


(a) Clustering with minimal total cut 4 and small normalized$^{\text{vert}}$ cut 0.256

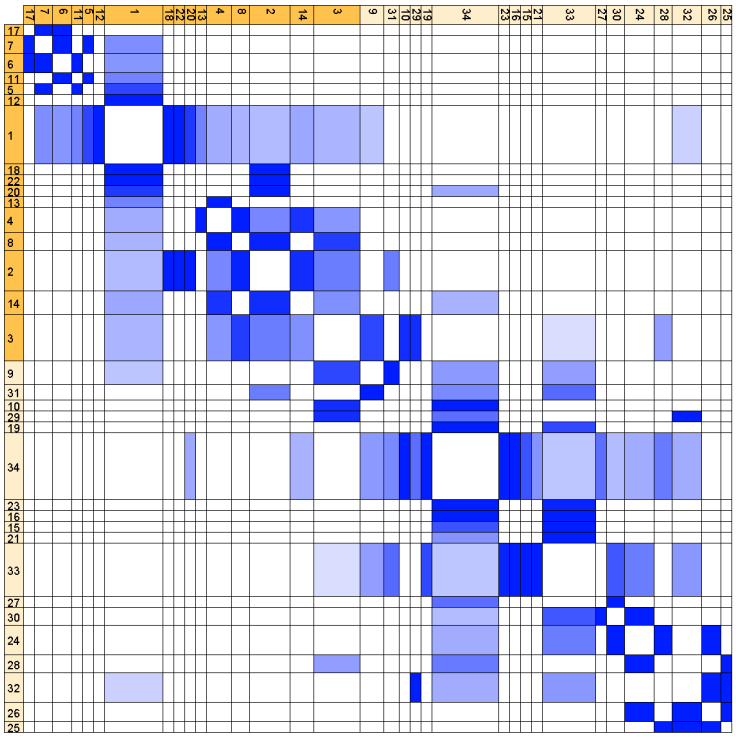


(b) Clustering with small normalized$^{\text{endv}}$ cut 0.335

Figure 4.5: Clusterings of the Southern Women graph

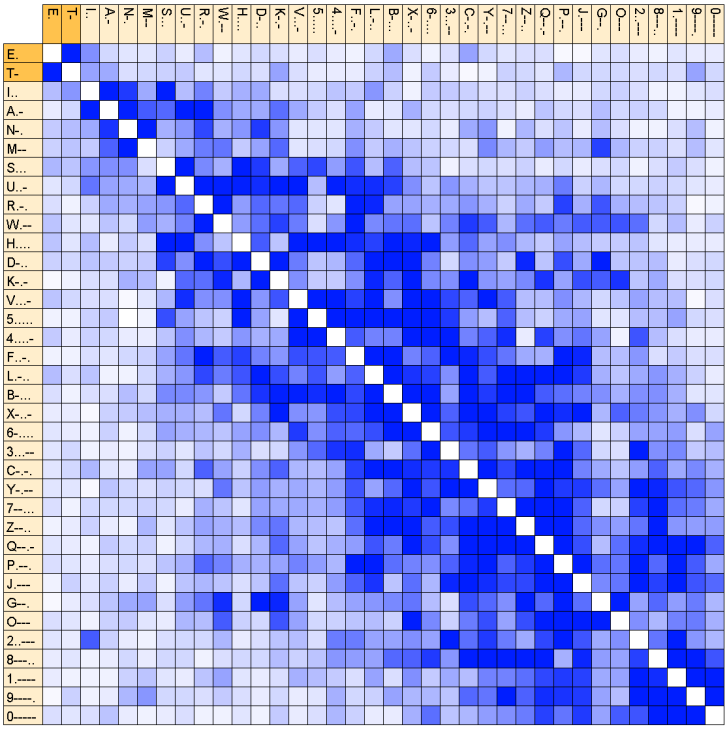


(a) Clustering with small normalized$^{\text{vert}}$ cut 0.186

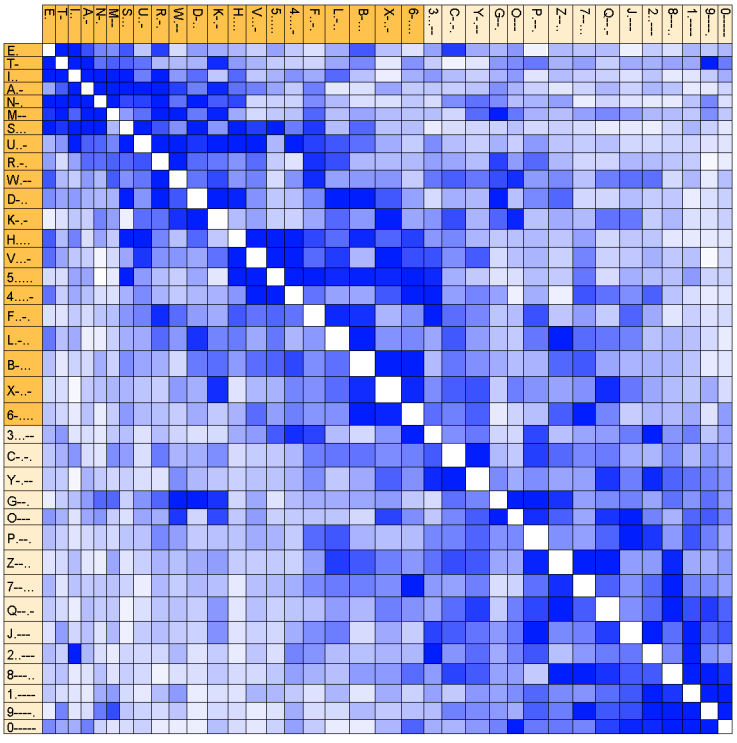


(b) Clustering with small normalized$^{\text{endv}}$ cut 0.181

Figure 4.6: Clusterings of the Karate Club graph. A clustering with minimal total cut 3 separates the vertex 19 from the remaining vertices.

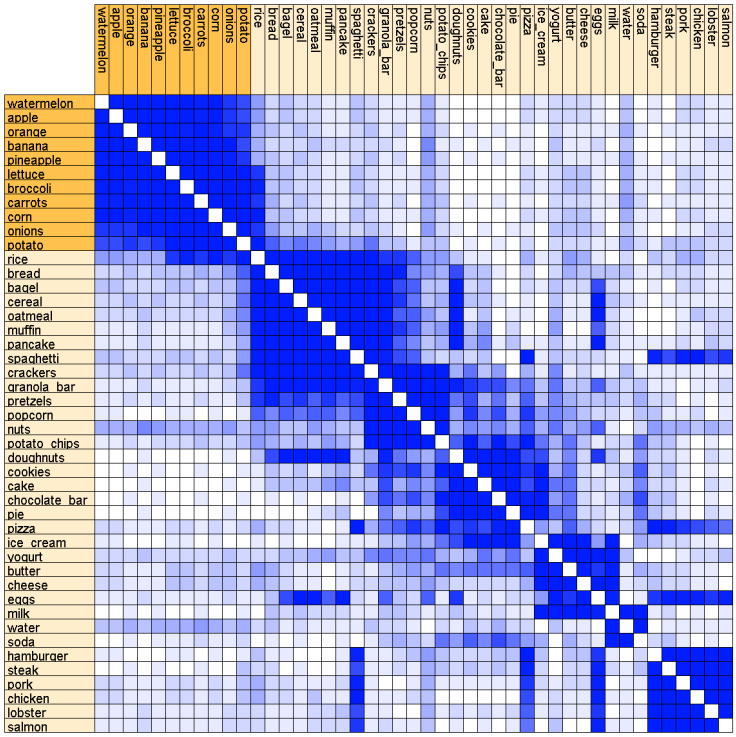


(a) Clustering with small normalized$^{\text{vert}}$ cut 0.299

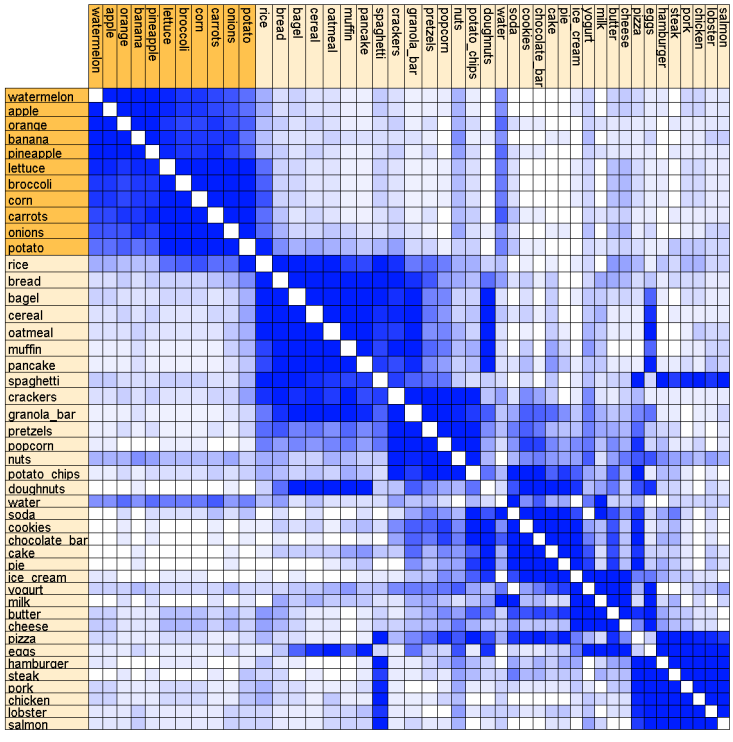


(b) Clustering with small normalized$^{\text{endv}}$ cut 0.690

Figure 4.7: Clusterings of the Morse Code Confusion graph. A clustering with minimal total cut 227 separates the vertex E. from the remaining vertices.

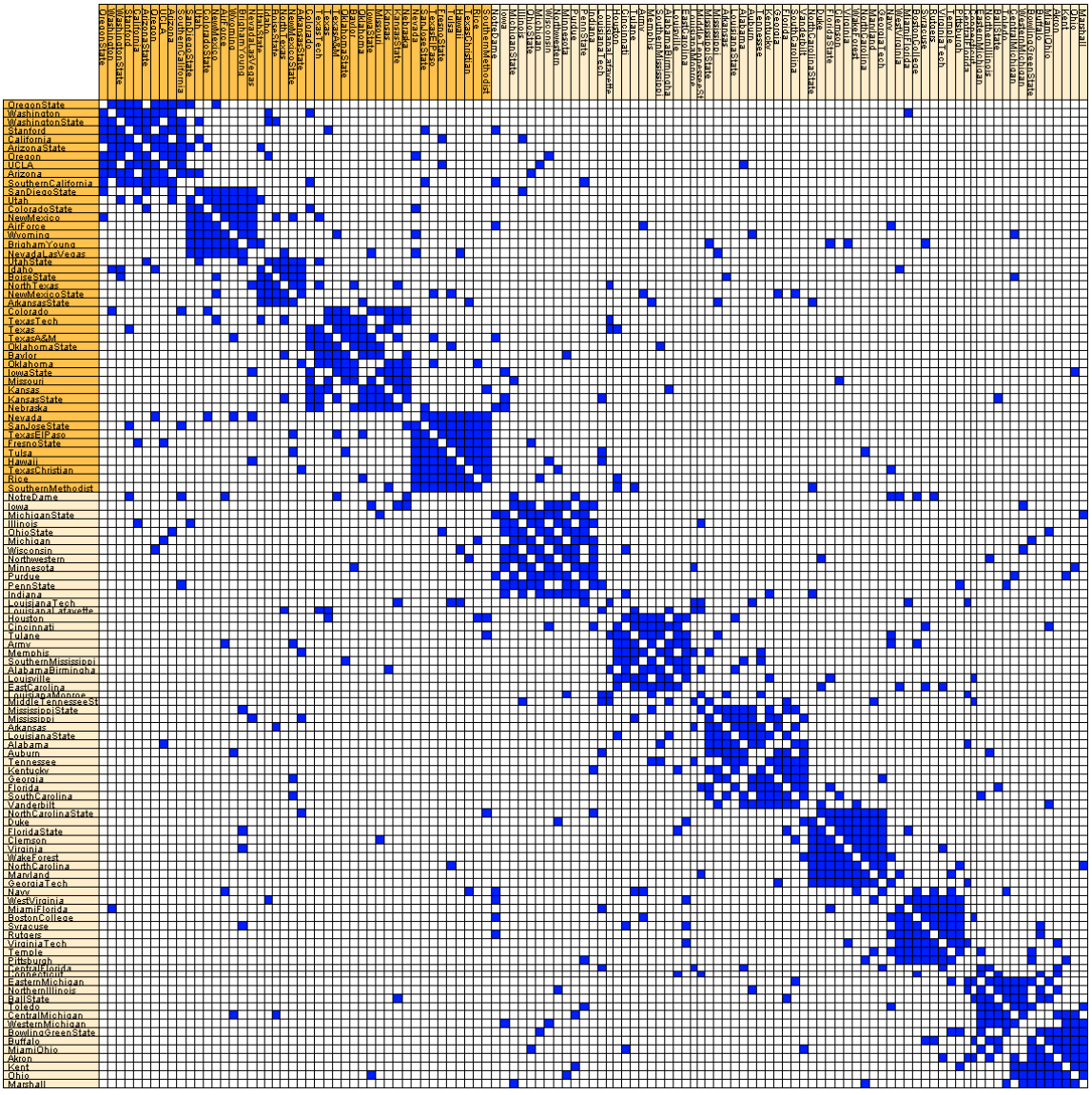


(b) Clustering with small normalized$^{\text{vert}}$ cut 0.311



(c) Clustering with small normalized$^{\text{endv}}$ cut 0.291

Figure 4.8: Clusterings of the Food Classification graph. A clustering with minimal total cut 134 separates the vertex water from the other vertices.



Clustering with small normalized$^{\text{vert}}$ cut 0.197 and small normalized$^{\text{endv}}$ cut 0.196

Figure 4.9: Clusterings of the College Football graph. A clustering with minimal total cut 7 separates the vertex Connecticut from the remaining vertices.

4.5 Summary

4.5.1 Main Results

- When the total atedge length, the scaled atedge length, and the normalized atedge length (defined in Section 3.1) are applied to graph clusterings, they are also called total (atedge) cut, scaled (atedge) cut, and normalized (atedge) cut. The total cut equals the number of inter-cluster atedges, the scaled cut equals the inter-cluster density, and the normalized cut equals the ratio of the inter-cluster density to the density of the graph.
- The measures are directly reflected in matrix visualizations of the cluster graph: The total cut equals the color weight above the diagonal elements, and the scaled cut equals the average color density above (or outside) the diagonal elements.
- The normalized cut rewards inter-cluster sparsity, and its optimal clusterings tend to be coarse-grained. An alternative, symmetric specialization of the normalized atedge length rewards intra-cluster density, and its optimal clusterings tend to be fine-grained.
- To remove the biases of an absolute measure (like the total atedge length), it can be divided through its value for uniform density (as in the normalized atedge length), or decreased by its value for uniform density (as in the shifted atedge length). The shifted atedge length has the advantage that its intra-cluster specialization is ranking-equivalent to its inter-cluster specialization, but it is scale-dependent and therefore unsuitable e.g. for graph layouts.
- Many existing clustering quality measures are biased. Of all surveyed multiway measures, only the shifted$^{\text{endv}}$ cut (originally proposed by Newman), the scaled$^{\text{endv}}$ cut sum (a generalized and corrected version of a measure by Shi and Malik), and the normalized$^{\text{endv}}$ cut are not biased for graphs with uniform density$^{\text{endv}}$. However, the scaled$^{\text{endv}}$ cut sum weights atedges between small clusters higher than atedges between large clusters, and the intra-cluster version of the normalized$^{\text{endv}}$ cut is not ranking-equivalent to the inter-cluster version.
- Empirical observations for several real-world graphs confirm the internal validity of the normalized$^{\text{vert}}$ cut and normalized$^{\text{endv}}$ cut, and the external validity primarily of the normalized$^{\text{endv}}$ cut, with the above-mentioned tendency towards coarse-grained clusterings as the only significant limitation.

4.5.2 Measures

Clustering quality measures for a base graph $G = (\mathcal{V}, \mathcal{E})$, a clustering p, and the resulting cluster graph $\mathrm{p}(G) = (\mathcal{V}', \mathcal{E}')$. The formulas are derived from the measures for assignments in Section 3.1 by substituting $\overline{\delta}$ for d as distance measure.

Total atedge cut (total atedge length):

$$\begin{aligned} \mathrm{Q}(\mathrm{p}) &= \sum\nolimits_{\{v_1,v_2\}\in\mathcal{E}:\mathrm{p}(v_1)\neq\mathrm{p}(v_2)} 1 \\ &= \sum\nolimits_{\{v'_1,v'_2\}\in\mathcal{E}':v'_1\neq v'_2} 1 \end{aligned}$$

- bias towards clusterings with few inter-cluster atvertex pairs (Theorem 3.1)
- bias towards graphs with small density (Theorem 3.1)

Scaled atedge cut (scaled atedge length):

$$\begin{aligned} \mathrm{Q}^{\mathrm{scal}}(\mathrm{p}) &= \frac{\sum_{\{v_1,v_2\}\in\mathcal{E}:\mathrm{p}(v_1)\neq\mathrm{p}(v_2)} 1}{\sum_{\{v_1,v_2\}\in\mathcal{V}^{\langle 2\rangle}:\mathrm{p}(v_1)\neq\mathrm{p}(v_2)} 1} \\ &= \frac{\sum_{\{v'_1,v'_2\}\in\mathcal{E}':v'_1\neq v'_2} 1}{\sum_{\{v'_1,v'_2\}\in\mathcal{V}'^{\langle 2\rangle}} 1} \end{aligned}$$

- no bias towards any clustering for graphs with uniform density (Theorem 3.1)
- bias towards graphs with small density (Theorem 3.1)

Normalized atedge cut (normalized atedge length):[13]

$$\begin{aligned} \mathrm{Q}^{\mathrm{norm}}(\mathrm{p}) &= \frac{\sum_{\{v_1,v_2\}\in\mathcal{E}:\mathrm{p}(v_1)\neq\mathrm{p}(v_2)} 1}{\sum_{\{v_1,v_2\}\in\mathcal{V}^{\langle 2\rangle}:\mathrm{p}(v_1)\neq\mathrm{p}(v_2)} 1} \Big/ \frac{|\mathcal{E}|}{|\mathcal{V}^{\langle 2\rangle}|} \\ &= \frac{\sum_{\{v'_1,v'_2\}\in\mathcal{E}':v'_1\neq v'_2} 1}{\sum_{\{v'_1,v'_2\}\in\mathcal{V}'^{\langle 2\rangle}} 1} \Big/ \frac{|\mathcal{E}'|}{|\mathcal{V}^{\langle 2\rangle}|} \end{aligned}$$

- no bias towards any clustering for graphs with uniform density (Theorem 3.1)
- no bias towards graphs with particular vertex weights or density (Theorem 3.1)

[13] The normalized atedge cut cannot be determined solely from the cluster graph because $|\mathcal{V}^{\langle 2\rangle}|$ depends on the weights of the individual vertices in the base graph.

Chapter 5

Quality Measures for Graph Orderings

Graph orderings are special graph assignments, and thus the quality measures for graph assignments from Chapter 3 are also quality measures for graph orderings. The first section of this chapter shows that the restricted positions in graph orderings enable simplifications of the measures, and even remove biases. The second section discusses related quality measures from the literature, and the third section presents example orderings of real-world graphs.

5.1 Simplified Definitions

Graph orderings are graph assignments with real numbers as positions, with the function $|p_1 - p_2|$ as distance measure for two positions p_1 and p_2, and with the additional restriction that the distance of neighboring vertices reflects their weight (see Section 2.3). The main theorem of this section states that because of these restrictions, all orderings of a given graph have the same total atvertex distance. This removes the respective bias of the total atedge length, and enables simplified formulations of the assignment quality measures from Chapter 3, which are summarized in Subsection 5.4.2 on page 96.

5.1.1 The Total Atedge Length

Visualization In matrix visualizations, each edge is represented by two matrix elements. The edge weight equals the color weight in each of the matrix elements. The edge length equals the distance of each of the two matrix elements to the diagonal of the matrix. The distance of a matrix element to the diagonal is measured as the distance from the center of the matrix element either to the center of the diagonal matrix element in the same row, or equivalently to the center of the diagonal matrix element in the same column (see Figure 5.1). (It may also be measured as $\sqrt{2}$ times the length of the perpendicular from the center of the matrix element to the imag-

inary diagonal line.) The total atedge length equals the *weighted* color weight in the matrix visualization, where the weighting is with respect to the distance to the diagonal. That is, the total atedge length is small if the color weight is small or close to the diagonal.

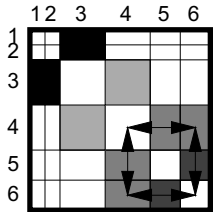
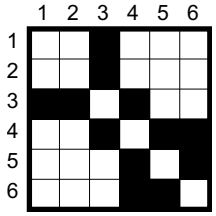
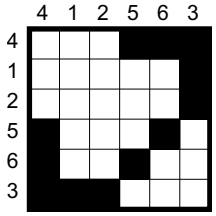


Figure 5.1: Representation of the length of the edge $\{4, 6\}$ in a matrix visualization

Examples The total atedge length of the orderings p_1 and p_2 in Figure 5.2 is 8 and 20, respectively. Accordingly, the color weight in the matrix representation of p_1 is closer to the diagonal than in the matrix representation of p_2.



Ordering p_1, with $\mathrm{Q}(\mathrm{p}_1) = 8$;
$\mathrm{Q}^{\mathrm{norm}}(\mathrm{p}_1) = \frac{8}{35}/\frac{6}{15} \approx 0.57$



Ordering p_2, with $\mathrm{Q}(\mathrm{p}_2) = 20$;
$\mathrm{Q}^{\mathrm{norm}}(\mathrm{p}_2) = \frac{20}{35}/\frac{6}{15} \approx 1.43$

Figure 5.2: Orderings of a graph with unit vertex weights

No Bias towards Orderings with Small Atvertex Distances According to Theorem 3.1, the total atedge length is biased towards assignments with small distances between the atvertices. Theorem 5.1 below states that this bias disappears for graph orderings, because all orderings of a graph have the same total atvertex distance (as shown in Lemma 5.2). This is not surprising for graphs with unit vertex weights, where the vertex positions are the same in all orderings (namely $\{0, 1, ..., |V|-1\}$), but it is less obvious for graphs with arbitrary vertex weights.

Theorem 5.1 *Let* $\mathcal{V} = (V, \mathrm{w})$ *be a nonempty multiset of atvertices, let* $m \in \mathbb{N}_+$*, and let* p *be an ordering of* $\mathcal{G}(\mathcal{V}, m)$*. Then the total atedge length* $\mathrm{Q}(\mathrm{p})$ *of* $\mathcal{G}(\mathcal{V}, m)$ *in* p *is*

$$\frac{m}{|\mathcal{V}^{\langle 2 \rangle}|} \cdot \frac{1}{6}\left(|\mathcal{V}|^3 - \textstyle\sum_{v \in V} \mathrm{w}(v)^3\right) .$$

Proof: Follows from Theorem 3.1 and Lemma 5.2. □

Lemma 5.2 *Let $G = ((V, \mathrm{w}), \mathcal{E})$ be a graph, and let* p *be an ordering of G. Then*

$$\sum_{\{v_1,v_2\}\in\mathcal{V}^{\langle 2\rangle}} |\mathrm{p}(v_1) - \mathrm{p}(v_2)| \;=\; \frac{1}{6}\left(|(V,\mathrm{w})|^3 - \sum_{v\in V} \mathrm{w}(v)^3\right) \;.$$

Proof: (For graphs with unit vertex weights.) A simpler proof for the special case of unit vertex weights is given before the more complicated proof for general vertex weights. By the definition of orderings, the vertices of a graph with unit vertex weights have the positions $0, 1, ..., |V|-1$. Thus

$$\begin{aligned}
\sum_{\{v_1,v_2\}\in\mathcal{V}^{\langle 2\rangle}} |\mathrm{p}(v_1) - \mathrm{p}(v_2)| &= \sum_{\{p_1,p_2\}\in\{0,...,|V|-1\}^{(2)}} |p_1 - p_2| \\
&= \sum_{p_1=0}^{|V|-1} \sum_{p_2=0}^{p_1} (p_1 - p_2) \\
&= \sum_{p_1=0}^{|V|-1} \sum_{p_2=0}^{p_1} p_2 \\
&= \sum_{p_1=0}^{|V|-1} \frac{1}{2}(p_1^2 + p_1) \\
&= \frac{1}{2}\sum_{p_1=0}^{|V|-1} p_1^2 + \frac{1}{2}\sum_{p_1=0}^{|V|-1} p_1 \\
&= \frac{1}{12}(2|V|^3 - 3|V|^2 + |V|) + \frac{1}{4}(|V|^2 - |V|) \\
&= \frac{1}{6}(|V|^3 - |V|)
\end{aligned}$$

□

Proof: (For graphs with general vertex weights.) The basic idea is that the total atvertex distance in an ordering p of G almost equals the total atvertex distance of an ordering p′ of a graph G' with $|(V, \mathrm{w})|$ vertices of weight 1. The total atvertex distance in p′ was shown to be $\frac{1}{6}(|(V,\mathrm{w})|^3 - |(V,\mathrm{w})|)$ in the above proof for unit vertex weights. The total atvertex distance in p is generally smaller, because the distance between each vertex v and itself in p is 0, while the total distance between the corresponding $\mathrm{w}(v)$ vertices in p′ is positive if $\mathrm{w}(v) > 1$. This total distance is $\frac{1}{6}(\mathrm{w}(v)^3 - \mathrm{w}(v))$ for each vertex v, and thus $\frac{1}{6}(\sum_{v\in V} \mathrm{w}(v)^3 - |(V,\mathrm{w})|)$ for the entire ordering.

For the formal derivation, let $n := |V|$ and, without loss of generality, let $V = \{v_1, ..., v_n\}$ with $\mathrm{p}(v_1) < ... < \mathrm{p}(v_n)$.

$$\begin{aligned}
& \sum_{\{v_1,v_2\}\in\mathcal{V}^{\langle 2\rangle}} |\mathrm{p}(v_1) - \mathrm{p}(v_2)| \\
= \; & \sum_{\{v_1,v_2\}\in V^{(2)}} \mathrm{w}(v_1)\mathrm{w}(v_2)|\mathrm{p}(v_1) - \mathrm{p}(v_2)| \\
= \; & \sum_{i=2}^{n} \sum_{j=1}^{i-1} \mathrm{w}(v_i)\mathrm{w}(v_j)(\mathrm{p}(v_i) - \mathrm{p}(v_j)) \\
= \; & \sum_{i=2}^{n} \sum_{j=1}^{i-1} \sum_{d=-(\mathrm{w}(v_j)-1)/2}^{+(\mathrm{w}(v_j)-1)/2} \mathrm{w}(v_i)(\mathrm{p}(v_i) - \mathrm{p}(v_j) - d) \\
= \; & \sum_{i=2}^{n} \sum_{q=\mathrm{p}(v_1)-(\mathrm{w}(v_1)-1)/2}^{\mathrm{p}(v_{i-1})+(\mathrm{w}(v_{i-1})-1)/2} \mathrm{w}(v_i)(\mathrm{p}(v_i) - q)
\end{aligned}$$

$$
\begin{aligned}
&= \sum_{i=2}^{n} \sum_{d=-(\mathrm{w}(v_i)-1)/2}^{+(\mathrm{w}(v_i)-1)/2} \sum_{q=\mathrm{p}(v_1)-(\mathrm{w}(v_1)-1)/2}^{\mathrm{p}(v_{i-1})+(\mathrm{w}(v_{i-1})-1)/2} (\mathrm{p}(v_i)+d-q) \\
&= \sum_{i=2}^{n} \sum_{d=-(\mathrm{w}(v_i)-1)/2}^{+(\mathrm{w}(v_i)-1)/2} \sum_{q=\mathrm{p}(v_1)-(\mathrm{w}(v_1)-1)/2}^{\mathrm{p}(v_i)+d-1} (\mathrm{p}(v_i)+d-q) \\
&\quad - \sum_{i=2}^{n} \sum_{d=-(\mathrm{w}(v_i)-1)/2}^{+(\mathrm{w}(v_i)-1)/2} \sum_{q=\mathrm{p}(v_i)-(\mathrm{w}(v_i)-1)/2}^{\mathrm{p}(v_i)+d-1} (\mathrm{p}(v_i)+d-q) \\
&= \sum_{i=1}^{n} \sum_{d=-(\mathrm{w}(v_i)-1)/2}^{+(\mathrm{w}(v_i)-1)/2} \sum_{q=\mathrm{p}(v_1)-(\mathrm{w}(v_1)-1)/2}^{\mathrm{p}(v_i)+d-1} (\mathrm{p}(v_i)+d-q) \\
&\quad - \sum_{i=1}^{n} \sum_{d=-(\mathrm{w}(v_i)-1)/2}^{+(\mathrm{w}(v_i)-1)/2} \sum_{q=\mathrm{p}(v_i)-(\mathrm{w}(v_i)-1)/2}^{\mathrm{p}(v_i)+d-1} (\mathrm{p}(v_i)+d-q) \\
&= \sum_{p=\mathrm{p}(v_1)-(\mathrm{w}(v_1)-1)/2}^{\mathrm{p}(v_n)+(\mathrm{w}(v_n)-1)/2} \sum_{q=\mathrm{p}(v_1)-(\mathrm{w}(v_1)-1)/2}^{p-1} (p-q) \\
&\quad - \sum_{i=1}^{n} \sum_{d=-(\mathrm{w}(v_i)-1)/2}^{+(\mathrm{w}(v_i)-1)/2} \sum_{q=-(\mathrm{w}(v_i)-1)/2}^{d-1} (d-q) \\
&= \sum_{p=0}^{\mathrm{p}(v_n)+(\mathrm{w}(v_n)-1)/2-\mathrm{p}(v_1)+(\mathrm{w}(v_1)-1)/2} \sum_{q=0}^{p-1} (p-q) \\
&\quad - \sum_{i=1}^{n} \sum_{d=0}^{\mathrm{w}(v_i)-1} \sum_{q=0}^{d-1} (d-q) \\
&= \sum_{p=0}^{|(V,\mathrm{w})|-1} \sum_{q=0}^{p-1} q \quad - \quad \sum_{i=1}^{n} \sum_{d=0}^{\mathrm{w}(v_i)-1} \sum_{q=0}^{d-1} q \\
&= \frac{1}{6}(|(V,\mathrm{w})|^3 - |(V,\mathrm{w})|) \quad - \quad \frac{1}{6} \sum_{v \in V} (\mathrm{w}(v)^3 - \mathrm{w}(v)) \\
&= \frac{1}{6} \left(|(V,\mathrm{w})|^3 - \sum_{v \in V} \mathrm{w}(v)^3 \right)
\end{aligned}
$$

□

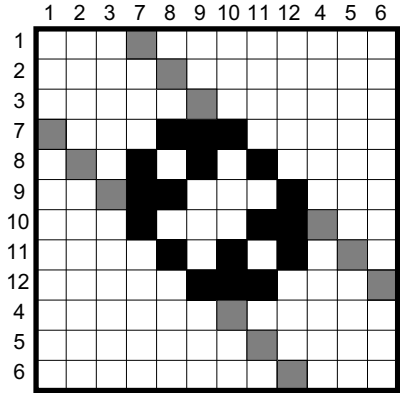
5.1.2 The Scaled and the Normalized Atedge Length

Simplified Definition The general definition of the scaled atedge length from Section 3.1 can be simplified using Lemma 5.2, as summarized in Subsection 5.4.2.

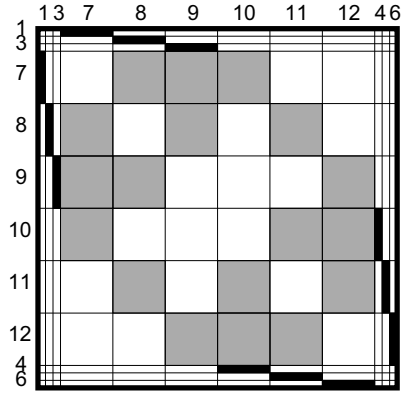
Both the scaled and the normalized atedge length differ from the total atedge length only by a constant factor for a given graph. Thus orderings with minimal normalized atedge length can be computed simply by minimizing the total atedge length. The scaled atedge length is somewhat redundant for orderings, because the total atedge length already is not biased towards any ordering; the normalized atedge length still has the additional benefit of being not biased towards particular graphs.

Visualization The remarks on the visualization of the total atedge length in the previous subsection apply similarly to the scaled and normalized atedge length, because both differ from the total atedge length only by a constant factor (for a fixed graph).

Examples In Figure 5.2, the ordering p_1 has a normalized atedge length smaller than 1, and thus the total atedge length is smaller than for uniformly distributed atedges. Similarly, the total atedge length in the ordering p_2 is larger than for uniformly distributed atedges.

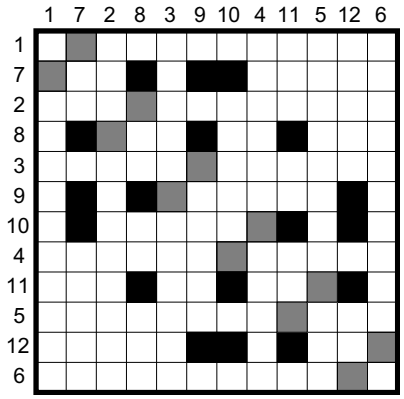


Ordering$^{\text{vert}}$ p_1,
with $Q(p_1) = 52$,
$Q^{\text{normvert}}(p_1) = \frac{52}{286} / \frac{24}{66} = 0.5$,
$Q^{\text{normendv}}(p_1) = \frac{52}{2938} / \frac{24}{1002} \approx 0.74$

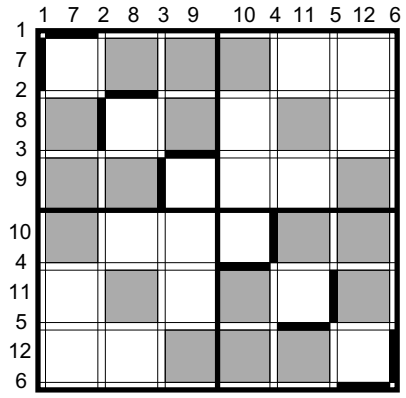


Ordering$^{\text{endv}}$ p_2,
with $Q(p_2) = 310$,
$Q^{\text{normvert}}(p_2) = \frac{310}{1468} / \frac{24}{66} \approx 0.58$,
$Q^{\text{normendv}}(p_2) = \frac{310}{18088} / \frac{24}{1002} \approx 0.72$

Orderings with optimal normalized$^{\text{vert}}$ atedge length



Ordering$^{\text{vert}}$ p_3,
with $Q(p_3) = 68$,
$Q^{\text{normvert}}(p_3) = \frac{68}{286} / \frac{24}{66} \approx 0.65$,
$Q^{\text{normendv}}(p_3) = \frac{68}{4090} / \frac{24}{1002} \approx 0.69$



Ordering$^{\text{endv}}$ p_4,
with $Q(p_4) = 290$,
$Q^{\text{normvert}}(p_4) = \frac{290}{1252} / \frac{24}{66} \approx 0.64$,
$Q^{\text{normendv}}(p_4) = \frac{290}{18088} / \frac{24}{1002} \approx 0.67$

Orderings with optimal normalized$^{\text{endv}}$ atedge length

Figure 5.3: Orderings of a graph without vertex weights

5.1.3 On Graphs without Vertex Weights

A graph without vertex weights can be considered either as graph with unit vertex weights or as graph with unit endvertex weights, but not as both at the same time. That is, an ordering$^{\text{endv}}$ should not be visualized in a matrix$^{\text{vert}}$, otherwise there may be gaps or overlaps between the rows and columns; and the quality of an ordering$^{\text{endv}}$ should not be measured with the scaled$^{\text{vert}}$ or normalized$^{\text{vert}}$ atedge length, otherwise the simplification of their denominator to $\frac{1}{6}(|V|^3 - |V|)$ is generally incorrect. Figure 5.3 (discussed below) makes an exception to enable a comparison of the normalized$^{\text{vert}}$ and the normalized$^{\text{endv}}$ atedge length on identical orderings.

The bias of the normalized$^{\text{vert}}$ atedge length towards high-degree vertices on central positions (Theorem 3.2), and the absence of this bias for the normalized$^{\text{endv}}$ atedge length (Theorems 3.1$^{\text{endv}}$, 5.1$^{\text{endv}}$), is illustrated by the four orderings in Figure 5.3. The normalized$^{\text{vert}}$ atedge length of the orderings in the upper row (which place the six high-degree vertices at the six central positions) is smaller than the normalized$^{\text{vert}}$ atedge length of the orderings in the lower row (which mix high-degree vertices and low-degree vertices). In contrast, the normalized$^{\text{endv}}$ atedge lengths in the lower row are smaller than in the upper row.

5.2 Related Work

Techniques for ordering objects have developed in various scientific fields under different names. In archeology, *seriation* determines the chronological order of ancient artifacts [Ken06]. In psychology, *unidimensional scaling* was originally developed to derive interval-scaled measures from the human judgments of similarities or dissimilarities ([BG97, Section 13.5], [CC01, Section 2.6])[1]. In sociology, *sociomatrix permutation* was the first method for the identification of cohesive subgroups in social networks [WF94, Section 7.10]. In scientific computing, *sparse matrix reordering* improves the performance of numerical algorithms ([GL81, Chapter 4], [Pis84, Chapter 4]). In information visualization, Bertin proposed the *reorderable matrix* as a tool for understanding and communicating data [Ber81, Ber01].[2] Even within graph theory, graph orderings have been given various names like linear layout, linear ordering, linear arrangement, or permutation. A recent survey of graph orderings by Diaz, Petit and Serna [DPS02] mainly takes the perspective of algorithms and complexity, but also mentions applications.

This section focuses on quality measures for graph orderings. This excludes algorithms for computing good orderings of a given graph, which are surveyed for various quality measures in [DPS02], and specifically for the total atedge length in [KH02, Pet03, SRB06].

5.2.1 Basic Measures

This section lists quality measures only for a graph without vertex weights $(V, \mathcal{E})$ and an ordering$^{\text{vert}}$ p, because most related literature does not consider vertex weights and orderings with nonuniform spacings between vertices. By the definition of orderings$^{\text{vert}}$, the positions of the vertices in p are 0, 1, ..., $|V|-1$.

The ordering quality measures are described in a uniform format with the items definition, references, and complexity. This is the same format as used in Section 4.3 for clustering quality measures, except that clustering-specific items are removed. As

[1] Typical results of unidimensional scaling are not strictly graph orderings, but general one-dimensional layouts where the positions are not restricted to 0, 1, ..., $|V|-1$.

[2] Bertin also motivates the present work when he states: "To 'understand' is to reach the global level and *to discover significant groups*." ([Ber01, p. 9], emphasis in the original).

the algorithmic aspects of all listed quality measures have been studied for several decades, the references focus on surveys, and on applications for reordering matrix visualizations or otherwise revealing dense subgraphs.

The value for graphs with uniform density is not given for each measure separately, because all measures have similar biases. The bias-related theorems can be adapted to all listed measures for orderings$^{\text{vert}}$, with the following results:

- No bias towards any ordering for graphs with uniform density$^{\text{vert}}$ (Theorem 5.1$^{\text{vert}}$).
- Bias towards orderings with high-degree vertices on central positions for graphs with uniform density$^{\text{endv}}$ (Theorem 3.2).
- Bias towards graphs with few vertices and atedges (Theorem 5.1$^{\text{vert}}$).

All listed quality measures for orderings can be seen as generalized sums of the lengths of all atedges. As a first generalization, let the *s-total atedge length* (for $s \in \mathbb{R} \cup \{\infty\}$, $s > 0$; also known as s-discrepancy [JM92]) be defined as

$$\left(\sum_{\{v_1,v_2\}\in\mathcal{E}} |\mathrm{p}(v_1) - \mathrm{p}(v_2)|^s\right)^{1/s} .$$

1-Total Atedge Length

Definition: $\sum_{\{v_1,v_2\}\in\mathcal{E}} |\mathrm{p}(v_1) - \mathrm{p}(v_2)|$

References: Total atedge length in Sections 3.1, 5.1. The corresponding minimization problem is surveyed as Minimum Linear Arrangement problem in [DPS02].

Orderings with small 1-total atedge length have been suggested as an aid for the identification of hierarchical clusterings in [HGR82], and apparently correspond to what is called "r-enumeration" in Pajek [BM06], a program for the analysis and visualization of large networks which also provides ordered matrix$^{\text{vert}}$ visualizations.

Complexity: Minimization NP-hard even for graphs with unit edge weights (Optimal Linear Arrangement [GJ79, GT42]). Approximable within $O(\log |V|)$ [RR98, RR04]. See also [ACG+99, Problem GT44], [DPS02].

Related: The normalized$^{\text{vert}}$ atedge length (defined in Section 3.1) differs from the 1-total atedge length only by a constant factor for all orderings of a fixed graph. In contrast to the 1-total atedge length, it is not biased towards graphs with a particular number of vertices or atedges (Theorem 5.1$^{\text{vert}}$).

2-Total Atedge Length

Definition: $\sqrt{\sum_{\{v_1,v_2\}\in\mathcal{E}} (\mathrm{p}(v_1) - \mathrm{p}(v_2))^2}$

The square root is sometimes omitted, which does not change the minima for a fixed graph.

References: The 2-total atedge length was proposed in early works on matrix reordering in social network analysis [Kat47], with the justification that the squared Euclidean distance of the matrix element in the i-th row and j-th column to the diagonal is $\frac{1}{2}(i-j)^2$. It is available as "inertia around diagonal" in the matrix reordering tool PermutMatrix [CP05], where it is motivated by the physical analogy to the moment of inertia of a matrix element around the diagonal.

In several works (e.g. [JM92, BPS95, DH04]), the 2-total atedge length plays a special role not because the resulting orderings are more useful than for other s-total atedge lengths, but for computational reasons: The eigenvector corresponding to the second smallest eigenvalue of the Laplacian is a non-trivial one-dimensional layout which minimizes the 2-total atedge length (see Subsection 3.2.2). However, this one-dimensional layout is not an ordering because the vertex positions are not restricted to 0, 1, ..., $|V|-1$. It can be transformed into an ordering by correcting the spacing of the vertices, but the resulting ordering may be far from optimal [JM92].

Complexity: Minimization NP-hard even for graphs with unit edge weights [GP97].

∞-Total Atedge Length

Definition: $\max_{\{v_1,v_2\}\in\mathcal{E}} |\mathrm{p}(v_1) - \mathrm{p}(v_2)|$

References: Surveyed as bandwidth in [DPS02]. The main application is the reordering of the rows and columns of large sparse matrices for efficient storage and manipulation (see e.g. [GL81, Chapter 4], [Pis84, Chapter 4]). Applied as quality measure for matrix visualizations in [BHR96a].

Complexity: Minimization NP-hard [GJ79, GT40]). Approximation algorithms exist for particular classes of graphs (see [ACG+99, Problem GT42], [DPS02]).

The s-total atedge length can be further generalized to the s_1-s_2-total atedge length by decomposing it into two generalized sums. The following measure is a well-known member of this class.

1-∞-Total Atedge Length

Definition: $\sum_{v_1\in V} \max_{\{v_1,v_2\}\in\mathcal{E}} |\mathrm{p}(v_1) - \mathrm{p}(v_2)|$
where the maximum of an empty set is 0.

References: Surveyed as profile in [DPS02]. Mainly applied in sparse matrix technology (see e.g. [GL81, Chapter 4], [Pis84, Chapter 4]), where it is also called envelope size, as it is the number of matrix elements in a part of the matrix called envelope. Used as quality measure for matrix visualizations in [BHR96a].

Complexity: Minimization NP-hard [GJ79, GT40]), approximable within $O(\log |V|)$ [RR98, RR04] (both via the equivalent Interval Graph Completion problem). See also [ACG+99, Problem GT39], [DPS02].

Further variations of the total atedge length can be generated by using alternative distance measures for vertex positions in orderings. In this work, the standard distance measure for two positions p_1 and p_2 in an ordering is $|p_1 - p_2|$ (see Section 2.3). Alternatively, the distance of two vertex positions p_1 and p_2 in an ordering$^{\text{vert}}$ could be defined to be 0 if $|p_1 - p_2| \leq 1$ and 1 otherwise[3]. That is, all vertex positions have the same distance 1 unless they are neighbors or identical. In general, minimizing the total atedge length with this notion of distance does not ensure that densely connected vertices are grouped, because shortening an atedge is not rewarded unless its endvertices become neighbors. Nevertheless, this ordering quality measure has been proposed in the literature, although it is usually motivated from a different perspective:

Total Binary Atedge Length

Definition: $\sum_{\{v_1,v_2\}\in\mathcal{E},|\mathrm{p}(v_1)-\mathrm{p}(v_2)|>1} 1$

For a fixed graph, minimizing this term is equivalent to maximizing $\sum_{\{v_1,v_2\}\in\mathcal{E},|\mathrm{p}(v_1)-\mathrm{p}(v_2)|=1} 1$.

References: Minimization of the latter term is the path variant of the Traveling Salesperson Problem (TSP) [GP02], which is better known in its tour variant ([GJ79, ND22], [ACG+99, ND32]), i.e. as minimization of

$$\sum_{\{v_1,v_2\}\in\mathcal{E},\ \mathrm{p}(v_2)=(\mathrm{p}(v_1)+1)\ \bmod\ |V|} 1 \ .$$

A heuristic that greedily optimizes the total binary atedge length is proposed in [Gel71] for the seriation of archeological objects, and is available as multiple-fragment heuristic in the tool PermutMatrix [CP05]. The measure is applied in [ABK98] for ordering the variables in visualizations of vectorial data e.g. with the parallel coordinates technique.

Complexity: The path variant of the TSP is NP-hard [PS76, ABK98], thus also the maximization of the total binary atedge length. The minimization of the total binary atedge length can be easily reduced to its maximization.

[3]Regarding the bias towards placing high-degree vertices at central positions, it is important to note that the centrality of positions in an ordering depends on the distance measure. With this distance measure, each of the positions 1, ..., $|V|-2$ has a total distance to the other vertices of $|V|-3$, and is thus more central than the positions 0 and $|V|-1$.

Discussion All quality measure for orderings in this subsection can be seen as variants of the total atedge length, and have similar biases. In general, the choice of the parameter s of the s-total atedge length (or of s_1 and s_2 in the s_1-s_2-total atedge length) is application dependent. However, the 1-total atedge length (i.e. the total atedge length as introduced in Sections 3.1 and 5.1) has some unique properties.

First, only the 1-total atedge length weights the length of each atedge equally. That is, if the length of an atedge e is increased by 1, then the 1-total atedge length increases by 1, while the increase of the other s-total atedge lengths depends on length of e.

Second, each ordering induces $|V|-1$ distinct clusterings with two clusters, where each vertex is assigned to the first cluster if its position exceeds a certain threshold, and to the second cluster otherwise. The s-total atedge length of the ordering coincides with the s-total atedge cut of these clusterings only for $s = 1$. Formally, for an ordering$^{\text{vert}}$ p and a natural number q, the *atedge cut of* p *at* q is defined as the number of atedges that start at or cross the position q, i.e. $\sum_{\{v_1,v_2\}\in\mathcal{E}:\,\mathrm{p}(v_1)\leq q<\mathrm{p}(v_2)} 1$. The *$s$-total atedge cut* (for $s \in \mathbb{R}\cup\{\infty\}$, $s>0$) of an ordering p is defined as the generalized sum of the atedge cuts at all positions, i.e.

$$\left(\sum_{q=0}^{|V|-1}\left(\sum_{\{v_1,v_2\}\in\mathcal{E}:\,\mathrm{p}(v_1)\leq q<\mathrm{p}(v_2)} 1\right)^s\right)^{1/s}.$$

It is easy to show that for orderings$^{\text{vert}}$, the 1-total atedge length equals the 1-total atedge cut (see e.g. [DPS02, Observation 2.1]), while the s-total atedge length generally differs from the s-total atedge cut for $s\neq 1$.

Of course, the s-total atedge cut is another class of quality measures for graph orderings. It seems that these measures (except for $s = 1$) have rarely been applied for reordering matrix visualizations or generally grouping densely connected vertices. In graph theory, the ∞-total atedge cut has been studied as cutwidth (see [DPS02] for a survey).

5.2.2 Extensions

Ordering Quality without Bias towards Central High-Degree Vertices The quality measures in the previous subsection are biased towards orderings$^{\text{vert}}$ with high-degree vertices on central positions for graphs with uniform density$^{\text{endv}}$. In contrast, the scaled$^{\text{endv}}$ atedge length (defined in Section 3.1) is not biased towards any ordering$^{\text{vert}}$ or any other assignment for graphs with uniform density$^{\text{endv}}$ (by Theorem 3.1$^{\text{endv}}$). In the notation of the previous subsection, it is defined as

$$\frac{\sum_{\{v_1,v_2\}\in\mathcal{E}} |\mathrm{p}(v_1)-\mathrm{p}(v_2)|}{\sum_{\{v_1,v_2\}\in V^{(2)}} \deg(v_1)\deg(v_2)|\mathrm{p}(v_1)-\mathrm{p}(v_2)|}.$$

Lemma 5.2$^{\text{endv}}$ shows that the denominator takes the same value for all orderings$^{\text{endv}}$ of a fixed graph, and thus even the total atedge length (i.e. the numerator) is not biased towards high-degree vertices on central positions for orderings$^{\text{endv}}$.

However, most literature on graph orderings deals either with orderings$^{\text{vert}}$ or with general one-dimensional layouts (without restrictions of the vertex positions), but not with orderings$^{\text{endv}}$.

Ding and He [DH04] propose the following quality measure $\tilde{J}_2$ for orderings$^{\text{vert}}$:

$$\tilde{J}_2(\mathrm{p}) \;:=\; \frac{\sum_{\{v_1,v_2\}\in\mathcal{E}}(\mathrm{p}(v_1)-\mathrm{p}(v_2))^2}{\sum_{v\in V}\deg(v)(\mathrm{p}(v)-\bar{\mathrm{p}})^2} \text{ , where } \bar{\mathrm{p}} := \frac{\sum_{v\in V}\deg(v)\mathrm{p}(v)}{\sum_{v\in V}\deg(v)} \;.$$

It can be shown (e.g. [Chu97, Equation 1.5]) that

$$\tilde{J}_2(\mathrm{p}) \;=\; |(V,\deg)|\frac{\sum_{\{v_1,v_2\}\in\mathcal{E}}(\mathrm{p}(v_1)-\mathrm{p}(v_2))^2}{\sum_{\{v_1,v_2\}\in V^{(2)}}\deg(v_1)\deg(v_2)(\mathrm{p}(v_1)-\mathrm{p}(v_2))^2} \;.$$

Thus Ding and He's measure $\tilde{J}_2$ differs from the scaled$^{\text{endv}}$ atedge length only in the constant factor (for a fixed graph) $|(V,\deg)|$, and in the distance measure for vertex positions, which is $(\mathrm{p}(v_1)-\mathrm{p}(v_2))^2$ instead of $|\mathrm{p}(v_1)-\mathrm{p}(v_2)|$. Theorem 3.1$^{\text{endv}}$ shows that it is not biased towards any ordering (including orderings$^{\text{vert}}$ with high-degree vertices on central positions) for graphs with uniform density$^{\text{endv}}$.

Ding and He empirically compare good orderings of their scaled$^{\text{endv}}$ measure $\tilde{J}_2$ with good orderings of a similar scaled$^{\text{vert}}$ measure called $\tilde{J}_1$, and explain the observed success of $\tilde{J}_2$ as follows: "... objects with large [degree] are more likely to be permuted towards the middle using the weighted constraints ... This is favorable, since objects with large [degree] are more likely to have more edges, and moving these objects towards middle decreases the distances among these similar objects ..." [DH04, Section 3]. So Ding and He argue that $\tilde{J}_2$ is better than $\tilde{J}_1$ because it introduces (or strengthens) a bias towards high-degree vertices on central positions – although it actually removes this bias. (Their motivation for introducing $\tilde{J}_2$ is that the resulting one-dimensional layouts have been successfully used in spectral clustering by Shi and Malik [SM00] and Ding et al. [DHZ+01].)

Ordering Quality Based on Derived Graphs Graphs without vertex weights can be represented as adjacency matrices (see Section 1.3.2). In an adjacency matrix, each vertex is represented by the vector of its edge weights to the other vertices. A proximity matrix can be derived from the adjacency matrix by computing, for each pair of vertices, the proximity of their vector representations, using any proximity measure for vectors. Now an ordering of the original graph can be obtained by minimizing any quality measure for orderings for the derived proximity matrix (instead of the original graph).

Bond energy as an instantiation of this framework is usually applied to permute the rows and columns of matrices in order to find dense groups [MSW72, Len74, AH90, AHS90]. For graphs, it corresponds to the total binary atedge length (defined above) applied to the following similarities between vertices:

$$\mathrm{s}(v_1,v_2) = \sum\nolimits_{v\in V}\mathrm{f}(\{v_1,v\})\mathrm{f}(\{v_2,v\}) \;.$$

The application of ordering quality measures to a derived proximity matrix instead of the original graph has significant disadvantages. It introduces additional arbitrariness, because a proximity measure has to be chosen in addition to an ordering quality measure, and it generally complicates the interpretation of the resulting orderings (and their quality) in terms of the original graph, because it is not directly derived from this graph.

Ordering Quality for Bipartite Graphs and Vectorial Data A graph is bipartite if its set of vertices can be decomposed into two nonempty disjoint subsets such that no two vertices within the same set are adjacent. In bipartite graphs, the two subsets of vertices may be ordered separately. (In matrix visualizations, one subset may be represented by the rows and the other subset by the columns.) Quality measures based on edge lengths or cuts (such as those listed above) are not applicable in this case, because there are no edges between vertices of the same subset. Alternatives include orderings based on derived (non-bipartite) graphs as sketched in the previous paragraph, or orderings based on edge crossings as proposed in [MS00, SM05].

Vectorial data can be considered as bipartite graph, with the objects and variables as vertices, and the values of the variables as edge weights.

Ordering Quality for Graphs with Clusterings An ordering of a graph can be constructed from a clustering by assigning vertices of the same cluster to contiguous positions. (For hierarchical clusterings, all subclusters of the same cluster are placed contiguously.) Matrix visualizations with such orderings facilitate the manual evaluation of the clustering, because the intra-cluster vertex pairs of each cluster (which should be connected by many atedges) correspond to a square along the diagonal of the matrix, and the inter-cluster vertex pairs (which should be connected by few atedges) correspond to the remaining area of the matrix. Such matrix visualizations have been proposed by many authors (e.g. [Sne57, Lin73, ESBB98, SG03]), and are also used in the present work (see Subsection 2.4.1).

If all vertices of the same cluster are assigned to contiguous positions, the order of the clusters and the order of the vertices within each cluster is still arbitrary, and may be determined by minimizing any of the above-mentioned quality measures for orderings. In other words, graph orderings can be computed by minimizing a quality measure subject to the constraint that all vertices of the same cluster (in a given clustering) are placed at contiguous positions. For hierarchical clusterings where each cluster has a bounded number of direct subclusters, this constraint enables polynomial-time algorithms for some problems that are NP-hard in the general case, e.g. for minimizing the total atedge length [BYEFN01] and the total binary atedge length [BJGJ01, BJDG$^+$03]. Other proposals of such orderings for matrix visualizations include [GW72] (using the total binary atedge length), and [MAY03] (using the total atedge length).

5.3 Examples

This section serves three main purposes. First, it illustrates the difference between the normalized$^{\text{vert}}$ atedge length (with its bias towards high-degree vertices on central positions) and the normalized$^{\text{endv}}$ atedge length, for orderings of real-world graphs. Second, it provides empirical evidence for the internal and external validity of the normalized$^{\text{vert}}$ atedge length and the normalized$^{\text{endv}}$ atedge length as ordering quality measures. At the same time, it compares (to a limited extent) empirically the validity of existing ordering quality measures with the validity of a newly introduced measure: The normalized$^{\text{vert}}$ atedge length applied to orderings$^{\text{vert}}$ is equivalent to the well-known total atedge length (Lemma 5.2), and is similar to other existing quality measures which are also generalized sums of atedge lengths (see Section 5.2), and have also been applied almost exclusively to orderings$^{\text{vert}}$. The normalized$^{\text{endv}}$ atedge length (or equivalently, the total atedge length applied to orderings$^{\text{endv}}$) was newly proposed as the first ordering quality measure which is unbiased for graphs with uniform density$^{\text{endv}}$.

Method All graphs in this section are described in Appendix A.1. For each graph, an ordering$^{\text{vert}}$ with small normalized$^{\text{vert}}$ atedge length and an ordering$^{\text{endv}}$ with small normalized$^{\text{endv}}$ atedge length are presented in a matrix$^{\text{vert}}$ and a matrix$^{\text{endv}}$ visualization, respectively. The orderings were computed with a multi-scale heuristic for minimizing the total atedge length (i.e. for the Minimum Linear Arrangement problem) similar to that of Koren and Harel [KH02]. This heuristic provides no guarantee that the computed orderings are optimal, and the optimality of the orderings is not assumed in the following.

Validation methods for graph assignments and assignment quality measures were introduced in Section 1.3. Internal validity of an ordering quality measure requires that in orderings with small measurement values, densely connected vertices have small distances, and sparsely connected vertices have large distances. This is checked using the matrix visualizations, where the density is represented by the color density of the matrix elements. External validity of an ordering quality measure requires that orderings with small measurement values are externally valid, i.e. resemble known authoritative orderings. This is checked by comparing the computed orderings with the authoritative groupings described in Appendix A.1.

Results The orderings with small normalized$^{\text{vert}}$ atedge length and with small normalized$^{\text{endv}}$ atedge length are shown in Figures 5.4 to 5.11 on pages 87 to 94.

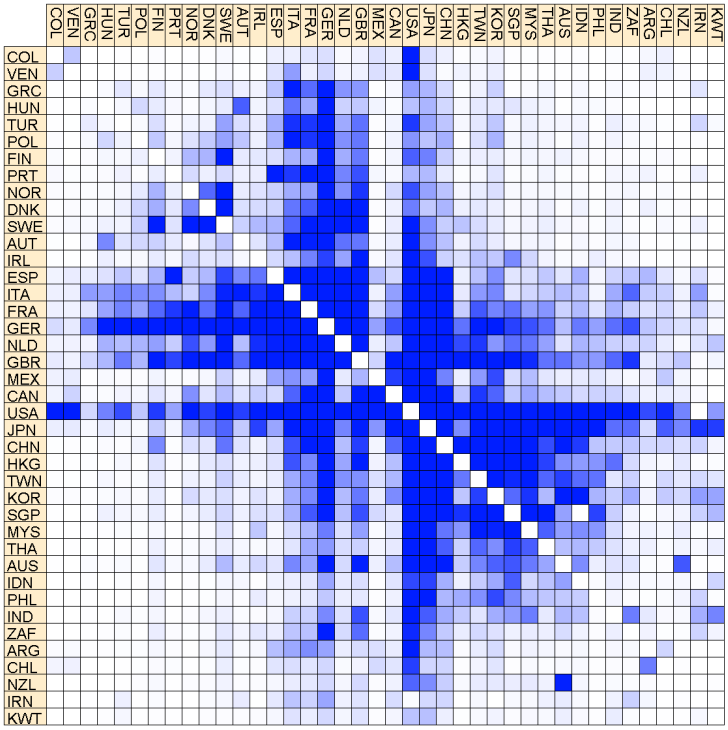
Concerning internal validity, the orderings with small normalized$^{\text{endv}}$ atedge length indeed group densely$^{\text{endv}}$ connected vertices and separate sparsely$^{\text{endv}}$ connected vertices for all examined graphs, as much as possible within the inherent limitations of graph orderings. This can be seen from the concentration of large color density near the diagonal, and of small color density in the top right and bottom left corner, in the respective matrix visualizations. The observations for the orderings with small normalized$^{\text{vert}}$ atedge length are similar.

Concerning external validity, the orderings with small normalized$^{\text{endv}}$ atedge length conform well to the authoritative groupings. The ordering of World Trade assigns contiguous positions not only to each main economical area (East Asia / Australia, America, and Europe) but also to some smaller groups of closely interlocked countries like HKG and CHN, AUS and NZL, GBR and IRL, the Nordic countries, and ESP and PRT. The ordering of US Airlines sorts the airports roughly from West to East, which reflects their relative geographical distances very well (given the restriction to one dimension and a fixed vertex spacing). The orderings of Southern Women and Karate Club not only place the two groups of the respective authoritative decompositions at successive positions, but even place the borderline cases (Pearl and person 9) at the positions between the groups. The authoritative grouping of Morse Code Confusion according to the number of beeps and the fraction of short beeps is two-dimensional, and thus difficult to represent as ordering. The ordering of Food Classification progresses from plant-based to animal-based, and assigns each major group of the authoritative decomposition (fruits, vegetables, grain-based, snacks, beverages, sweets, dairies, meat including seafood) to successive positions. Similarly, the ordering of College Football places the teams of each conference at contiguous positions.

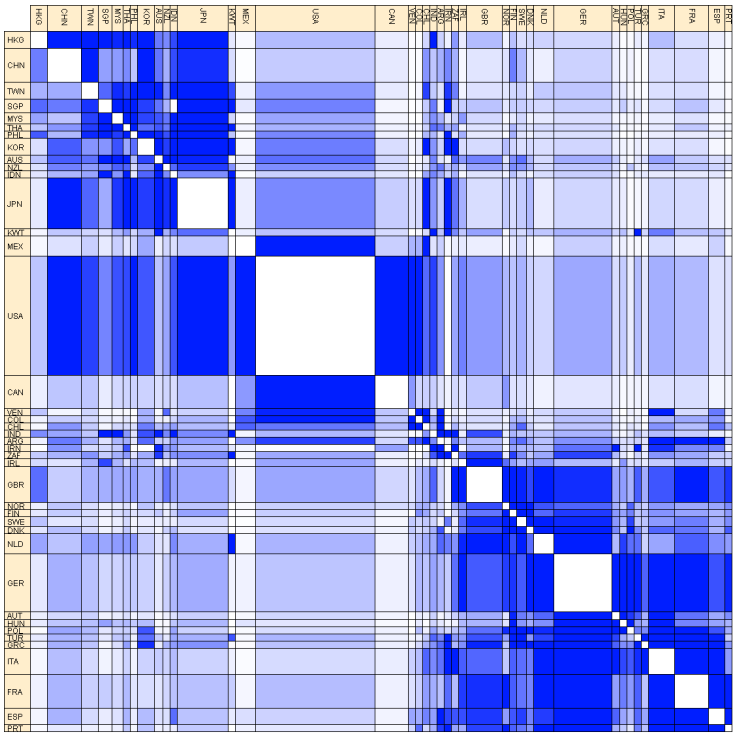
The orderings with small normalized$^{\text{vert}}$ atedge length are similar or identical to the orderings with small normalized$^{\text{endv}}$ atedge length for all examined graphs except World Trade and US Airlines, where the vertex degrees are extremely nonuniform. In both cases, the ordering with small normalized$^{\text{vert}}$ atedge length does not conform to the authoritative grouping, but mainly reflects the vertex degrees, with high-degree vertices in the center and low-degree vertices at the periphery.

In summary, the examined orderings with small normalized$^{\text{endv}}$ atedge length conform very well to the authoritative groupings, given the inherent limitations to one dimension and to a fixed spacing of vertices. The orderings with small normalized$^{\text{vert}}$ atedge length strongly differ from the corresponding orderings with small normalized$^{\text{endv}}$ atedge length only for the graphs with the most nonuniform vertex degrees, where they mainly reflect the degrees (as predicted by Theorem 3.2).

Because of their limitations to one dimension and to a fixed vertex spacing, orderings alone cannot represent most nontrivial vertex groupings with reasonable precision. However, an appropriate ordering of the vertices in matrix visualizations is essential to make groups and outliers apparent, as demonstrated in Figures 5.9 and 5.10 for the Food Classification graph.

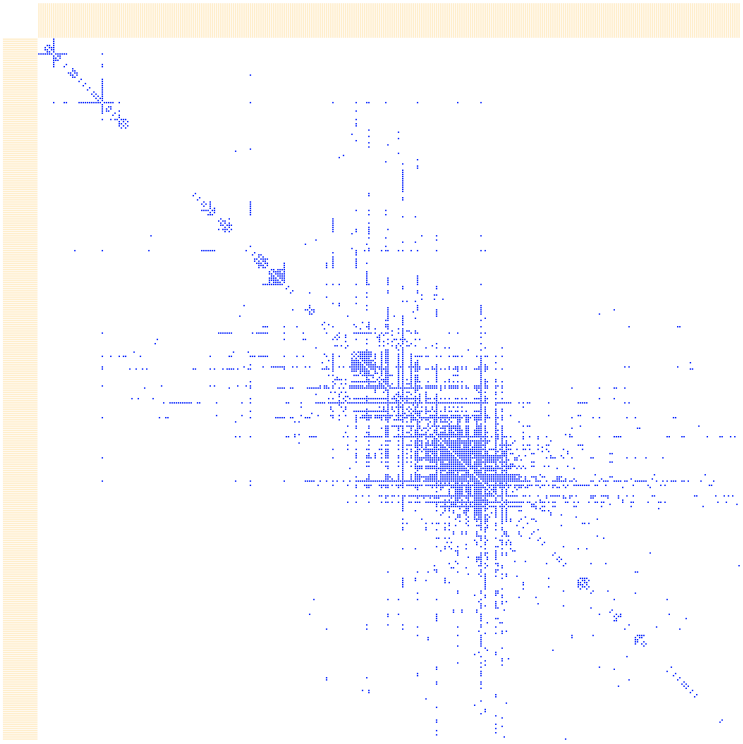


(a) Ordering$^{\text{vert}}$ with small normalized$^{\text{vert}}$ atedge length 0.374

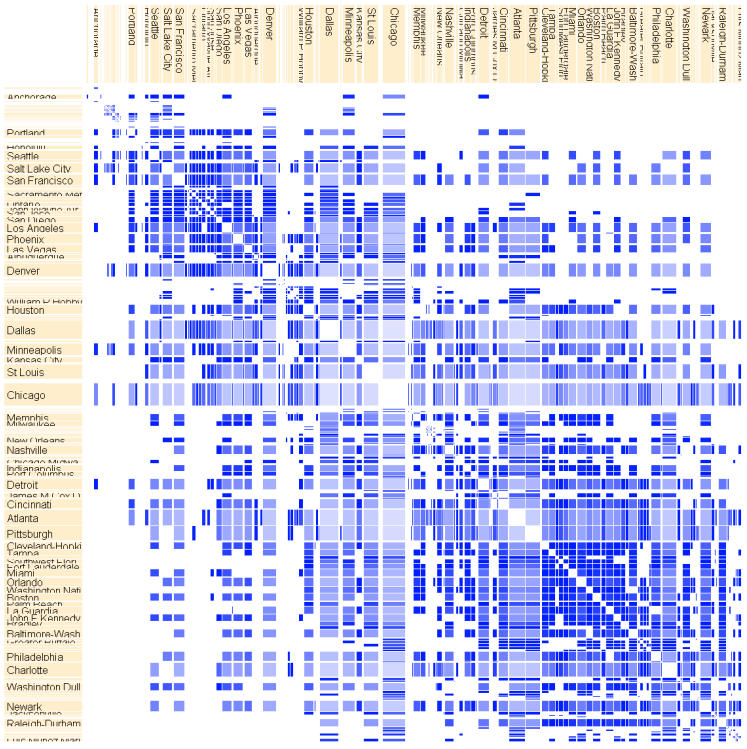


(b) Ordering$^{\text{endv}}$ with small normalized$^{\text{endv}}$ atedge length 0.624

Figure 5.4: Orderings of the World Trade graph

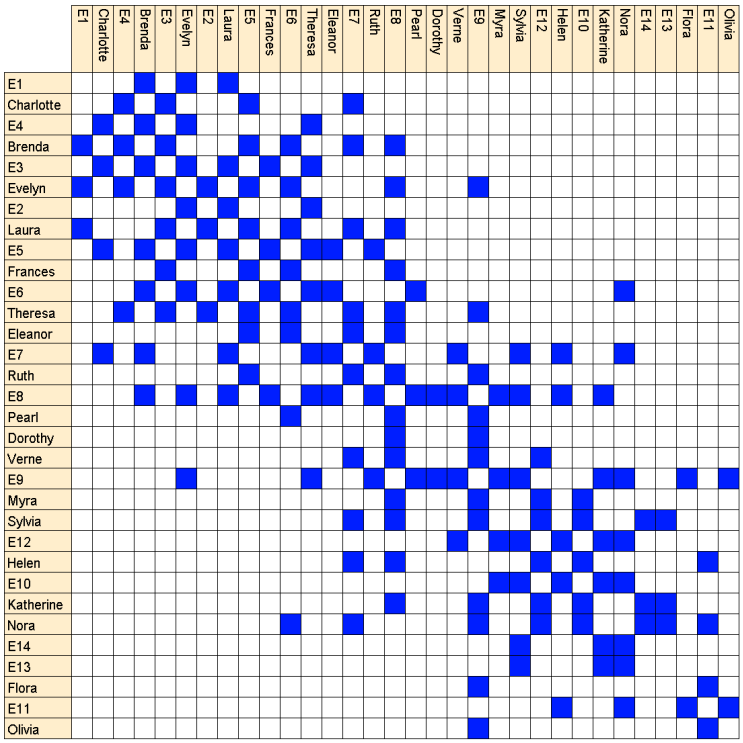


(a) Ordering$^{\text{vert}}$ with small normalized$^{\text{vert}}$ atedge length 0.247

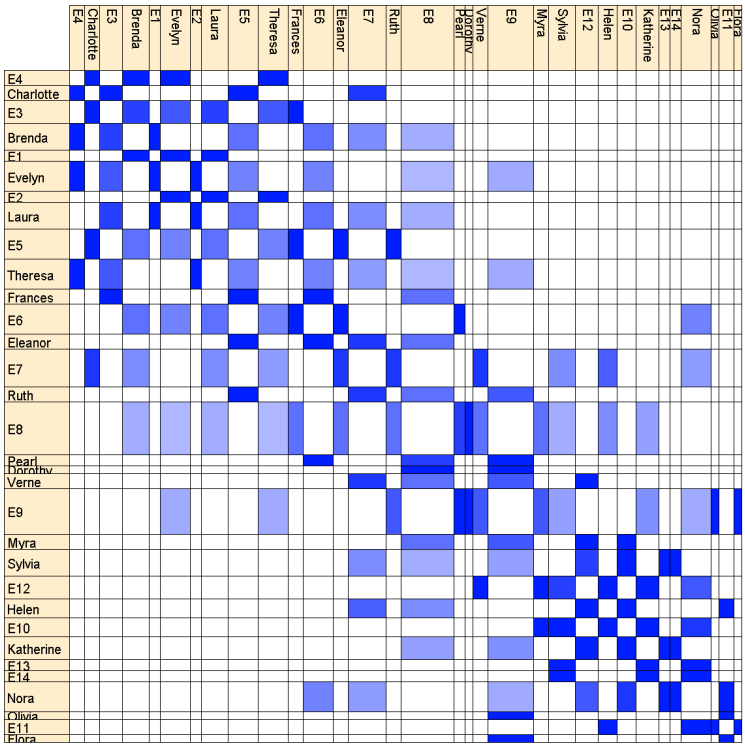


(b) Ordering$^{\text{endv}}$ with small normalized$^{\text{endv}}$ atedge length 0.444

Figure 5.5: Orderings of the US Airlines graph

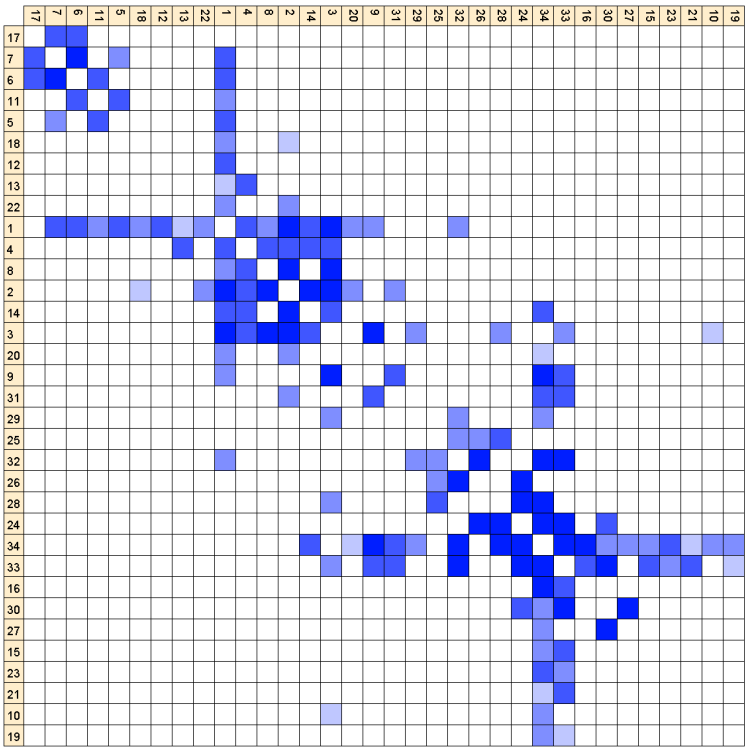


(a) Ordering$^{\text{vert}}$ with small normalized$^{\text{vert}}$ atedge length 0.414

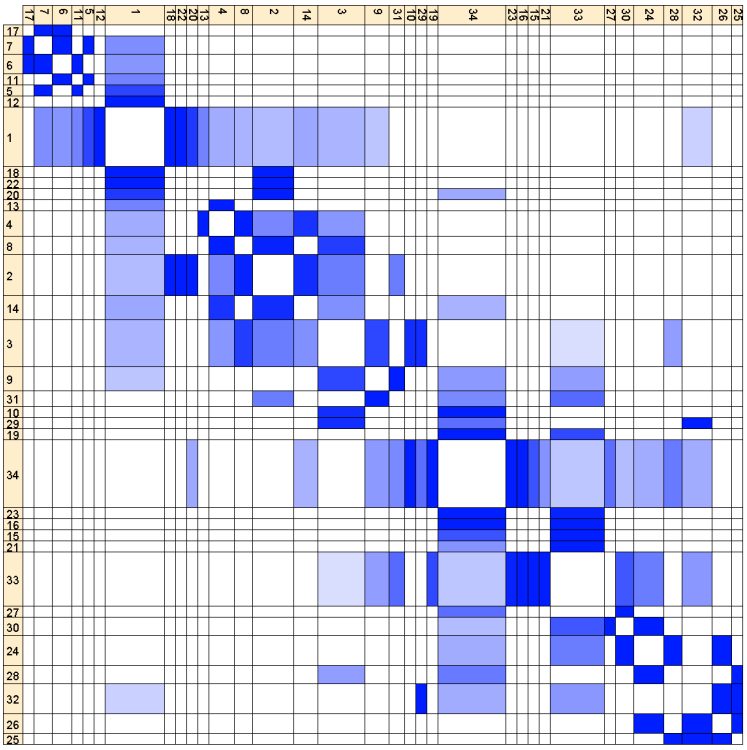


(b) Ordering$^{\text{endv}}$ with small normalized$^{\text{endv}}$ atedge length 0.446

Figure 5.6: Orderings of the Southern Women graph

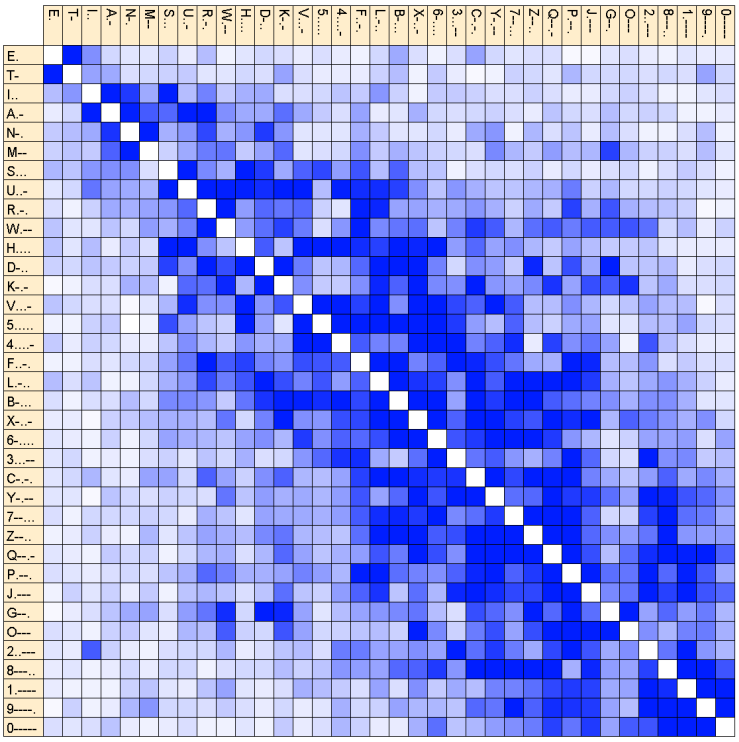


(a) Ordering$^{\text{vert}}$ with small normalized$^{\text{vert}}$ atedge length 0.309

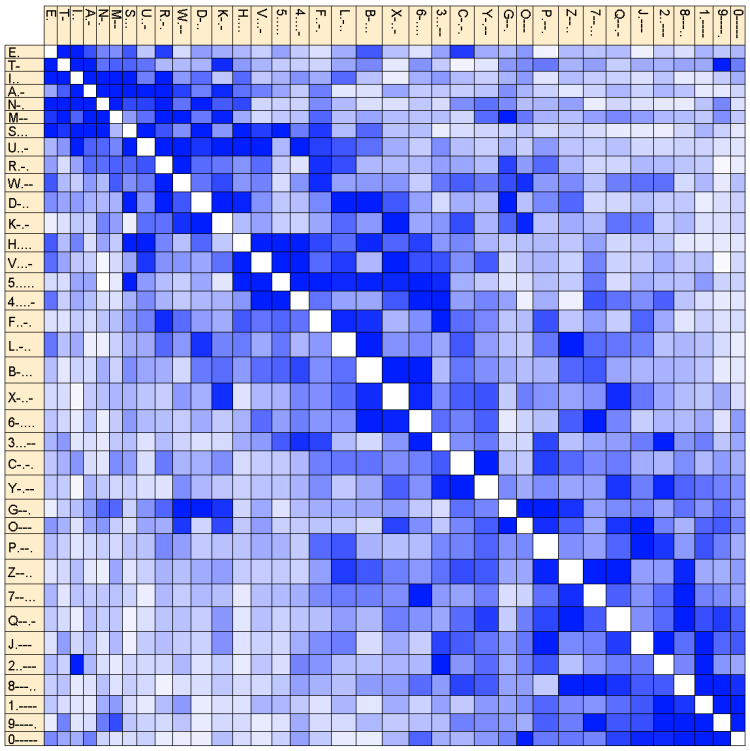


(b) Ordering$^{\text{endv}}$ with small normalized$^{\text{endv}}$ atedge length 0.348

Figure 5.7: Orderings of the Karate Club graph

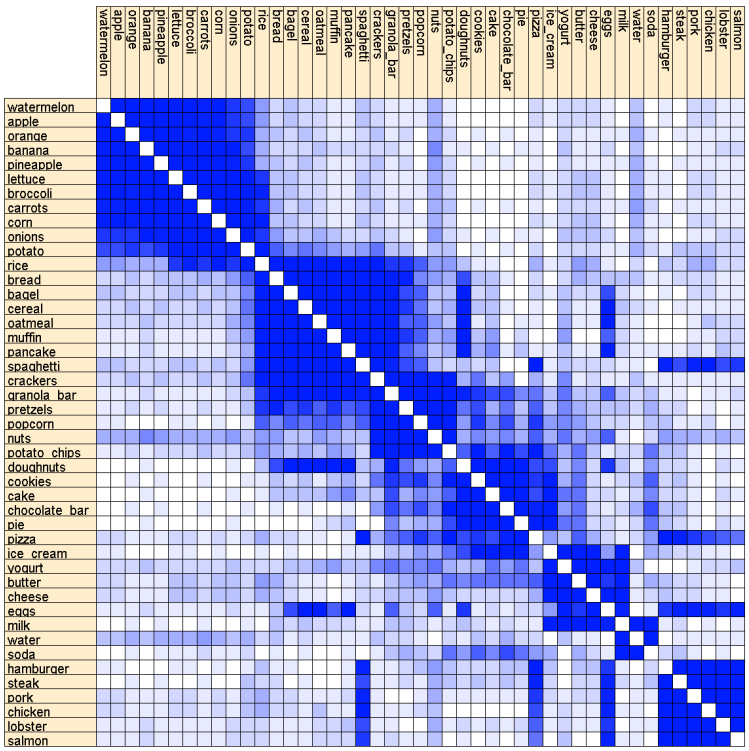


(a) Ordering$^{\text{vert}}$ with small normalized$^{\text{vert}}$ atedge length 0.647

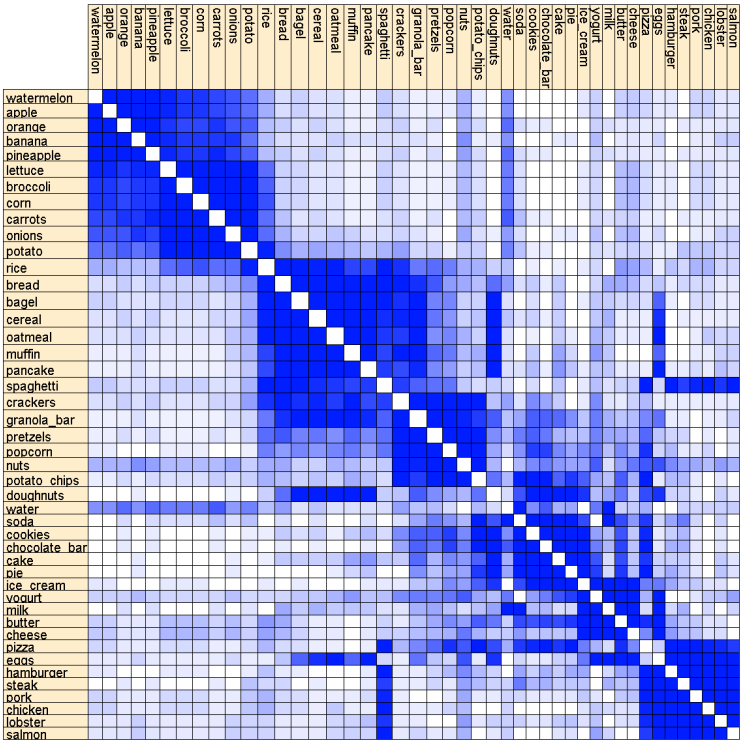


(b) Ordering$^{\text{endv}}$ with small normalized$^{\text{endv}}$ atedge length 0.721

Figure 5.8: Orderings of the Morse Code Confusion graph

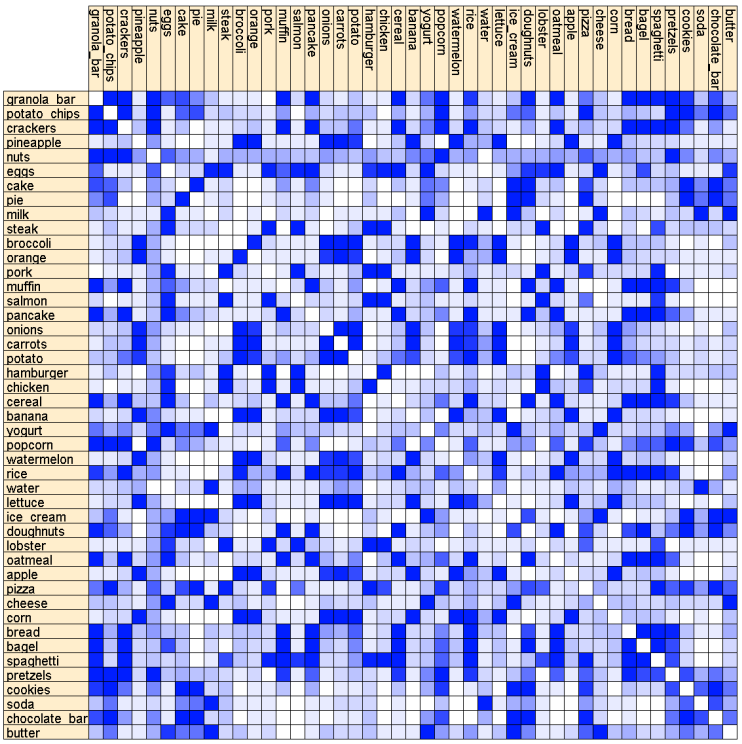


(a) Ordering$^{\text{vert}}$ with small normalized$^{\text{vert}}$ atedge length 0.451

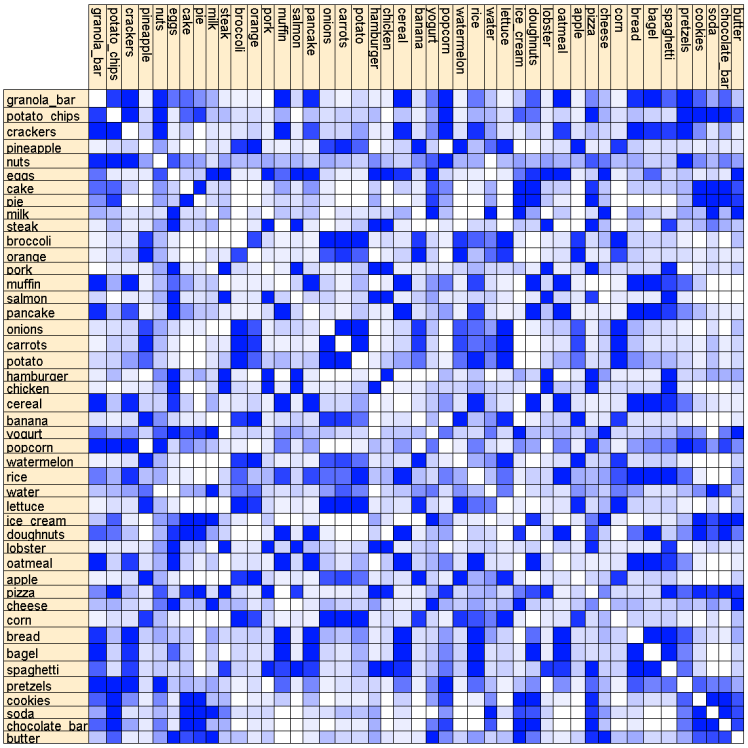


(b) Ordering$^{\text{endv}}$ with small normalized$^{\text{endv}}$ atedge length 0.462

Figure 5.9: Orderings of the Food Classification graph



(a) Pseudo-random ordering$^{\text{vert}}$ with normalized$^{\text{vert}}$ atedge length 1.001



(b) Pseudo-random ordering$^{\text{endv}}$ with normalized$^{\text{endv}}$ atedge length 1.004

Figure 5.10: Pseudo-random orderings of the Food Classification graph

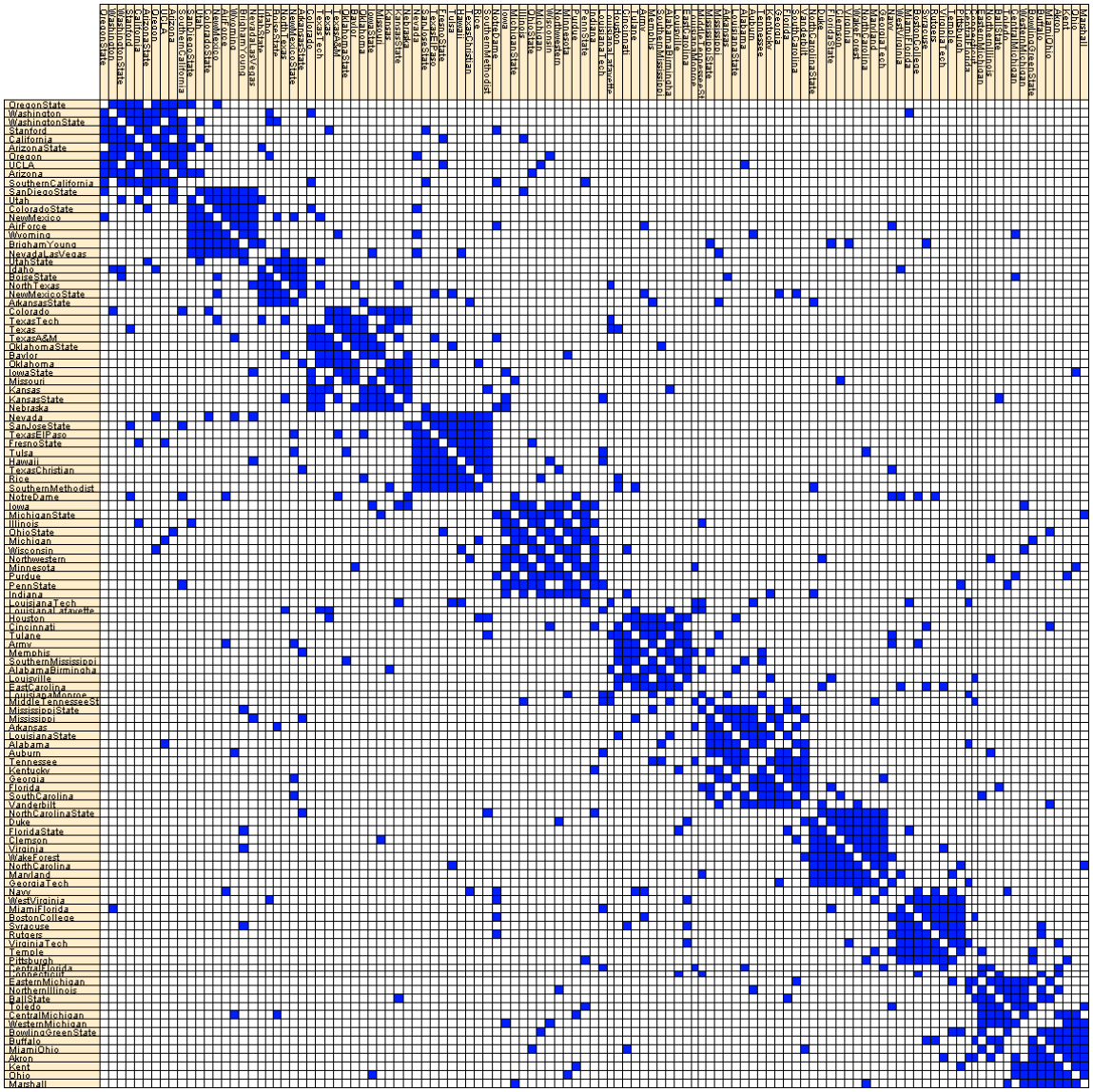


Figure 5.11: Ordering^{endv} of the College Football graph with small normalized^{endv} atedge length 0.276. Because of the almost uniform vertex degrees, the ordering quality and the matrix visualization for unit vertex weights are very similar.

5.4 Summary

5.4.1 Main Results

- All orderings of a graph have the same total atvertex distance (Lemma 5.2). Thus, even the total atedge length is not biased towards any ordering for graphs with uniform density (Theorem 5.1), and orderings with a minimal normalized atedge length can be found by minimizing the (simpler) total atedge length.
- The total atedge length of an ordering is reflected in the ordered matrix visualization of the graph by the color weight and its distance to the diagonal.
- Many ordering quality measures in the literature are generalized sums of the atedge lengths. Among those measures, the total atedge length has the unique properties of weighting the length of each atedge equally, and of being equivalent to the total atedge cut.
- The surveyed ordering quality measures for graphs without vertex weights are not biased towards any assignment for graphs with uniform density$^{\text{vert}}$, but biased towards placing high-degree vertices on central positions for graphs with uniform density$^{\text{endv}}$ (with an exception discussed in the first paragraph of Subsection 5.2.2). This bias can be avoided with scaled$^{\text{endv}}$ quality measures like the normalized$^{\text{endv}}$ atedge length, or with orderings$^{\text{endv}}$ (instead of orderings$^{\text{vert}}$).
- Empirical observations for several real-world graphs confirm the internal validity of the normalized$^{\text{vert}}$ atedge length and the normalized$^{\text{endv}}$ atedge length. The external validity of the (new) normalized$^{\text{endv}}$ atedge length can be significantly better than the external validity of the normalized$^{\text{vert}}$ atedge length (which is equivalent or similar to existing measures) for graphs with nonuniform vertex degrees.

5.4.2 Measures

Ordering quality measures for a graph $G = ((V, \mathrm{w}), \mathcal{E})$ and an ordering p. The formulas are derived from the definitions in Section 3.1 by substituting $|p_1 - p_2|$ for $\mathrm{d}(p_1, p_2)$ as distance measure, and by applying Lemma 5.2.

Total atedge length:
$$\mathrm{Q}(\mathrm{p}) = \sum\nolimits_{\{v_1,v_2\}\in\mathcal{E}} |\mathrm{p}(v_1) - \mathrm{p}(v_2)|$$

- no (!) bias towards any ordering for graphs with uniform density (Theorem 5.1)
- bias towards graphs with few atvertices and few atedges (Theorem 5.1)

Scaled atedge length:
$$\mathrm{Q}^{\mathrm{scal}}(\mathrm{p}) = \frac{\sum_{\{v_1,v_2\}\in\mathcal{E}} |\mathrm{p}(v_1) - \mathrm{p}(v_2)|}{\frac{1}{6}\left(|(V, \mathrm{w})|^3 - \sum_{v\in V} \mathrm{w}(v)^3\right)}$$

- no bias towards any ordering for graphs with uniform density (Theorem 5.1)
- bias towards graphs with small density (Theorem 5.1)

Normalized atedge length:
$$\mathrm{Q}^{\mathrm{norm}}(\mathrm{p}) = \mathrm{Q}^{\mathrm{scal}}(\mathrm{p}) \,\Big/\, \frac{|\mathcal{E}|}{|(V, \mathrm{w})^{\langle 2\rangle}|}$$

- no bias towards any ordering for graphs with uniform density (Theorem 5.1)
- no bias towards graphs with a particular number of atvertices or atedges (Theorem 5.1)

Chapter 6

Quality Measures for Graph Layouts

This chapter applies the quality measures for graph assignments from Chapter 3 to graph layouts. The first section starts with the observation that layouts with an optimal normalized atedge length tend to place vertices of a dense subgraph not only close to each other, but on the same position. It proposes a generalization of the normalized atedge length that allows to control this tendency. The second section shows that minima of the generalized quality measure correspond to equilibria of certain systems of attractive and repulsive forces between the vertices, which enables the computation of good layouts with existing force calculation algorithms. The third section proves another property of these equilibrium layouts, namely that the distance of subgraphs is closely related to their inter-density. The final sections discuss related quality measures and present example layouts for real-world graphs.

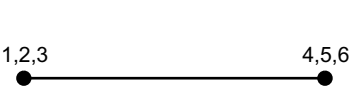
6.1 Generalized Definitions

Graph layouts are graph assignments where the positions are vectors of real numbers, and where the distance between positions is measured with the Euclidean norm (see Section 2.3). This section generalizes the quality measures for assignments from Section 3.1 such that very small and very large distances can be penalized.

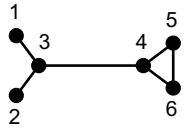
6.1.1 The Total Atedge Length

Visualization In box-line visualizations, the total atedge length equals the total area of the lines representing the edges (except lines representing loops, i.e. edges from a vertex to itself).

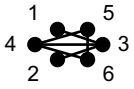
Examples All three layouts in Figure 6.1 have the same total atedge length. This illustrates that the total atedge length in itself is not a good indicator of layout quality. In fact, any layout (with positive total atedge length) can be transformed into a layout with any nonnegative total atedge length by simple scaling, i.e. by multiplying the positions with an appropriate constant factor.



Layout p_1 with
$Q^{norm}(p_1) \approx 0.28$



Layout p_2 with
$Q^{norm}(p_2) \approx 0.49$



Layout p_3 with
$Q^{norm}(p_3) \approx 1.32$

Figure 6.1: Layouts of a graph with unit vertex weights and unit edge weights

6.1.2 The r-Scaled Atedge Length

The graph in Figure 6.1 has two relatively dense subgraphs, namely the subgraph induced by the vertices 1, 2, 3, and the subgraph induced by the vertices 4, 5, 6. The left layout in Figure 6.1 has the optimal scaled atedge length among all layouts of this graph. Indeed, the layout groups the vertices of each dense subgraph very clearly by placing them at the same position. However, such radical grouping has also disadvantages for some application scenarios: The layouts do not reflect the structure within the groups (i.e. even denser subgraphs within the dense subgraphs), and the corresponding box-line visualization does not show the edges within the groups (because these edges have length 0).

This motivates the definition of a more general quality measure, with a parameter that controls how closely dense subgraphs are grouped and how distantly sparsely connected subgraphs are placed. Technically, this can be achieved by replacing the sum of the atvertex distances in the denominator of the scaled atedge length with a generalized sum that emphasizes small distances and deemphasizes large distances.

Definition The *r-scaled atedge length* (for $r \in \mathbb{R}$, $r \neq 0$) of a graph $G = (\mathcal{V}, \mathcal{E})$ in a layout p is

$$Q^{r\text{-scal}}(p) := \frac{Q(p)}{\left(\sum_{\{v_1,v_2\} \in \mathcal{V}^{\langle 2 \rangle}} \|p(v_1) - p(v_2)\|^r\right)^{1/r}}$$

The range of the r-scaled atedge length is the set $\mathbb{R} \cup \{+\infty\}$. The special case of $r = 1$ corresponds to the (ungeneralized) scaled atedge length. Smaller values of r correspond to less radical grouping, thus only the case of $r \leq 1$ is of interest in the following. The exclusion of $r = 0$ appears somewhat unnatural, and indeed this case will be allowed in the r-normalized atedge length defined in the next subsection.

The r-scaled atedge length is introduced only for graph layouts, because this generalization does not appear to be useful for the other types of graph assignments considered in this work, namely for graph clusterings and graph orderings. In clusterings, there is no reason to avoid that vertices are assigned to the same position, because the positions correspond to clusters. In orderings, assignments of different vertices to the same position are forbidden by definition, and the distance of two neighboring vertices v_1 and v_2 is fixed to $\frac{1}{2}(w(v_1) + w(v_2))$.

Bias The adaption of Theorem 3.1 from the total atedge length to the r-scaled atedge length is given below as Theorem 6.1. It states that the r-scaled atedge length is biased towards graphs with small density (like the scaled atedge length), and towards layouts p with a small ratio of the *total atvertex distance* $\sum_{\{v_1,v_2\}\in\mathcal{V}^{\langle 2\rangle}} \|\mathrm{p}(v_1)-\mathrm{p}(v_2)\|$ to the *r-total atvertex distance* $\left(\sum_{\{v_1,v_2\}\in\mathcal{V}^{\langle 2\rangle}} \|\mathrm{p}(v_1)-\mathrm{p}(v_2)\|^r\right)^{1/r}$ (while the scaled atedge length is not biased towards any layout). For $r<1$, the ratio of the total atvertex distance to the r-total atvertex distance is minimal if all pairs of vertices $\{v_1,v_2\}\in\mathcal{V}^{\langle 2\rangle}$ have the same distance [Bul03, Chapter III.3, Theorem 1]. Thus the r-total atvertex distance is biased towards assignments with uniform atvertex distances for $r<1$. In other words, it is biased *against* very small and very large atvertex distances – which was precisely the purpose of its definition.

Theorem 6.1 *Let $\mathcal{V}$ be a nonempty multiset of atvertices, let $m\in\mathbb{N}_+$, and let* p *be an assignment of $\mathcal{G}(\mathcal{V},m)$. Then the r-scaled atedge length* $\mathrm{Q}^{r\text{-scal}}(\mathrm{p})$ *of $\mathcal{G}(\mathcal{V},m)$ in* p *is*

$$\frac{m}{|\mathcal{V}^{\langle 2\rangle}|}\cdot\frac{\sum_{\{v_1,v_2\}\in\mathcal{V}^{\langle 2\rangle}} \|\mathrm{p}(v_1)-\mathrm{p}(v_2)\|}{\left(\sum_{\{v_1,v_2\}\in\mathcal{V}^{\langle 2\rangle}} \|\mathrm{p}(v_1)-\mathrm{p}(v_2)\|^r\right)^{1/r}}$$

Proof: Follows from Theorem 3.1. □

Using the notation of Theorem 6.1 and assuming that all pairs of different vertices have the same distance d in the layout p, the r-scaled atedge length of $\mathcal{G}(\mathcal{V},m)$ is

$$\frac{m}{|\mathcal{V}^{\langle 2\rangle}|}\cdot\frac{\sum_{\{v_1,v_2\}\in\mathcal{V}^{\langle 2\rangle}} d}{\left(\sum_{\{v_1,v_2\}\in\mathcal{V}^{\langle 2\rangle}} d^r\right)^{1/r}} = \frac{m}{|\mathcal{V}^{\langle 2\rangle}|}\cdot\frac{|\mathcal{V}^{\langle 2\rangle}|d}{(|\mathcal{V}^{\langle 2\rangle}|d^r)^{1/r}} = \frac{m}{|\mathcal{V}^{\langle 2\rangle}|^{1/r}} \tag{6.1}$$

Thus the r-scaled atedge length is biased towards graphs with few atedges and many atvertices if the distances are fixed, which motivates its normalization in the next subsection.

6.1.3 The r-Normalized Atedge Length

Definition Equation (6.1) (below Theorem 6.1) motivates a normalization of the r-scaled atedge length to remove its bias towards graphs with few atedges and many atvertices. The *r-normalized atedge length* (for $r\in\mathbb{R}$, $r\neq 0$) of a graph $G=(\mathcal{V},\mathcal{E})$ in a layout p is

$$\begin{aligned}\mathrm{Q}^{r\text{-norm}}(\mathrm{p}) &:= \mathrm{Q}^{r\text{-scal}}(\mathrm{p}) \Big/ \frac{|\mathcal{E}|}{|\mathcal{V}^{\langle 2\rangle}|^{1/r}} \\ &= \frac{\sum_{\{v_1,v_2\}\in\mathcal{E}} \|\mathrm{p}(v_1)-\mathrm{p}(v_2)\|}{|\mathcal{E}|} \Big/ \left(\frac{\sum_{\{v_1,v_2\}\in\mathcal{V}^{\langle 2\rangle}} \|\mathrm{p}(v_1)-\mathrm{p}(v_2)\|^r}{|\mathcal{V}^{\langle 2\rangle}|}\right)^{1/r}.\end{aligned}$$

The second formulation has a particularly simple interpretation: The r-normalized atedge length is the quotient of the arithmetic mean of the atedge lengths and the

r-th power mean of the atvertex distances. The 1-st and (-1)-st power mean are also known as arithmetic mean and harmonic mean, respectively. The definition extends naturally to the case of $r = 0$ by setting $\mathrm{Q}^{0\text{-norm}}(\mathrm{p}) := \lim_{r \to 0} \mathrm{Q}^{r\text{-norm}}(\mathrm{p})$, which is equivalent to [Bul03, Chapter III.1, Theorem 2]

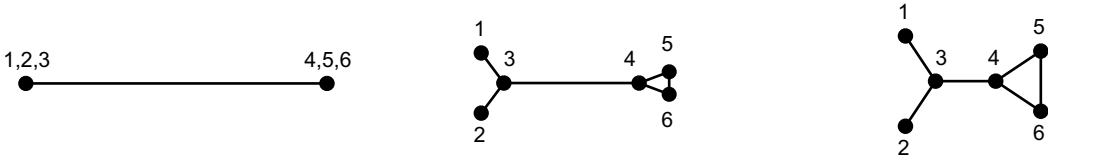
$$\mathrm{Q}^{0\text{-norm}}(\mathrm{p}) \;=\; \frac{\sum_{\{v_1,v_2\}\in\mathcal{E}} \|\mathrm{p}(v_1)-\mathrm{p}(v_2)\|}{|\mathcal{E}|} \;\Big/\; \exp\left(\frac{\sum_{\{v_1,v_2\}\in\mathcal{V}^{\langle 2\rangle}} \ln\|\mathrm{p}(v_1)-\mathrm{p}(v_2)\|}{|\mathcal{V}^{\langle 2\rangle}|}\right) .$$

The divisor is the geometric mean of the atvertex distances.

According to Theorem 6.1, the r-normalized atedge length in any layout of a graph with uniform density equals the quotient of the arithmetic mean and the r-th power mean of the atvertex distances. For $r < 1$, this quotient is greater than or equal to 1, with equality if and only if all pairs of different vertices have the same distance [Bul03, Chapter III.3, Theorem 1]. So the r-normalized atedge length is (like the r-scaled atedge length) biased against very small and very large distances between different vertices for $r < 1$. This is desired, it was the motivation for the generalization of the (1-)normalized atedge length to the r-normalized atedge length.

Examples In Figure 6.1, the 1-normalized atedge length of the left layout is much smaller than 1, thus the total atedge length in this layout is much smaller than for uniformly distributed atedges. The right layout has a 1-normalized atedge length greater than 1. The reason for these different 1-normalized atedge lengths despite equal total atedge lengths are the different total atvertex distances in the two layouts.

In Figure 6.2, the left layout is optimal with respect to the 1-normalized atedge length, and places vertices of the same dense subgraph at the same position. The middle layout is approximately optimal with respect to the 0-normalized atedge length, and still places vertices of the same dense subgraph fairly close to each other. The right layout is approximately optimal with respect to the -2-normalized atedge length, and distributes the vertices fairly uniformly, such that the dense subgraphs are hardly reflected in the layout. (Here it is important to distinguish the layout from the box-line visualization. The dense subgraphs may still recognizable in the visualization through the representations of the edges.)



Layout p_1 with
$\mathrm{Q}^{1\text{-norm}}(\mathrm{p}_1) \approx 0.28$,
$\mathrm{Q}^{0\text{-norm}}(\mathrm{p}_1) = +\infty$,
$\mathrm{Q}^{-2\text{-norm}}(\mathrm{p}_1) = +\infty$

Layout p_2 with
$\mathrm{Q}^{1\text{-norm}}(\mathrm{p}_2) \approx 0.42$,
$\mathrm{Q}^{0\text{-norm}}(\mathrm{p}_2) \approx 0.54$,
$\mathrm{Q}^{-2\text{-norm}}(\mathrm{p}_2) \approx 0.98$

Layout p_3 with
$\mathrm{Q}^{1\text{-norm}}(\mathrm{p}_3) \approx 0.59$,
$\mathrm{Q}^{0\text{-norm}}(\mathrm{p}_3) \approx 0.64$,
$\mathrm{Q}^{-2\text{-norm}}(\mathrm{p}_3) \approx 0.74$

Figure 6.2: Layouts of a graph with unit vertex weights and unit edge weights

6.1.4 On Graphs without Vertex Weights

The bias of the normalized$^{\text{vert}}$ atedge length towards high-degree vertices on central positions (Theorem 3.2), and the absence of this bias for the normalized$^{\text{endv}}$ atedge length (Theorem 3.1$^{\text{endv}}$), carry over to the generalized versions of these measures. This is illustrated by the six layouts in Figure 6.3. The three layouts in the upper row are approximately optimal with respect to the r-normalized$^{\text{vert}}$ atedge length, and tend to centralize high degree vertices. In the three layouts in the lower row, which are approximately optimal with respect to the r-normalized$^{\text{endv}}$ atedge length, this tendency is absent (for $r=1$) or much weaker. While the upper layouts mainly separate high-degree vertices from low-degree vertices, the lower layouts separate two relatively weakly connected subgraphs which both contain vertices of different degrees. This parallels the observations for orderings in Subsection 5.1.3 and for clusterings in Subsection 4.1.3. The examples also show that for $r < 1$, the bias towards central high-degree vertices interferes with the bias towards uniform vertex distances. The latter bias becomes more dominant with decreasing r, and the layouts with optimal r-normalized$^{\text{vert}}$ atedge length converge with the layouts with optimal r-normalized$^{\text{endv}}$ atedge length for decreasing r.

6.2 Formulation as r-LinPoly Energy Model

Energy-based (or force-directed) graph layout methods consist of two parts ([Bra01], [DETT99, Chapter 10]): The *energy model* specifies what is considered as a good layout, by mapping each layout to a real number (called its *total energy*) such that lower numbers are assigned to better layouts. The *energy minimization algorithm* searches a layout of a given graph with minimum total energy.

The r-normalized atedge length is already an energy model in this general sense. However, most popular energy minimization algorithms are limited to energy models of a particular form, where the total energy is the sum of energies between pairs of vertices. This section defines such an energy model and shows its equivalence to the r-normalized atedge length.

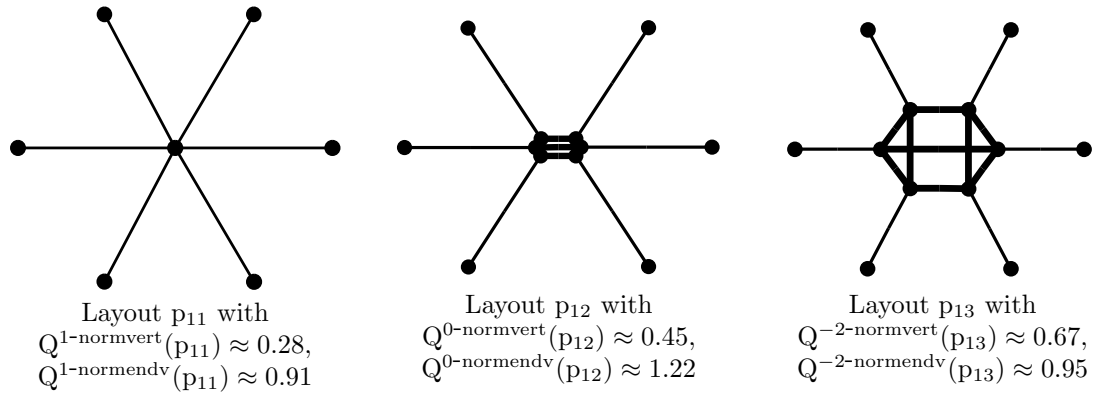
Definition The *r-LinPoly energy* (for $r \in \mathbb{R}$) of a graph $(\mathcal{V}, \mathcal{E})$ in a layout p is

$$\mathrm{U}^r(\mathrm{p}) \;:=\; \sum\nolimits_{\{v_1,v_2\}\in\mathcal{E}} \|\mathrm{p}(v_1)-\mathrm{p}(v_2)\| \;-\; \sum\nolimits_{\{v_1,v_2\}\in\mathcal{V}^{\langle 2\rangle}} \frac{1}{r}\|\mathrm{p}(v_1)-\mathrm{p}(v_2)\|^r \;.$$

for $r \neq 0$ and

$$\mathrm{U}^0(\mathrm{p}) \;:=\; \sum\nolimits_{\{v_1,v_2\}\in\mathcal{E}} \|\mathrm{p}(v_1)-\mathrm{p}(v_2)\| \;-\; \sum\nolimits_{\{v_1,v_2\}\in\mathcal{V}^{\langle 2\rangle}} \ln\|\mathrm{p}(v_1)-\mathrm{p}(v_2)\| \;.$$

The 0-LinPoly energy is also called *LinLog energy*. The range of the r-LinPoly energy is the set $\mathbb{R}\cup\{+\infty,-\infty\}$. The first term of the energy model can be interpreted as attraction between adjacent vertices, the second term as repulsion between different vertices.

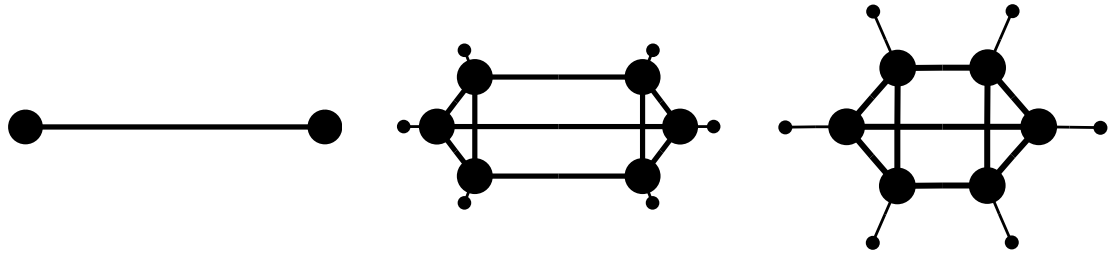


Layout p_{11} with $Q^{1\text{-normvert}}(p_{11}) \approx 0.28$, $Q^{1\text{-normendv}}(p_{11}) \approx 0.91$

Layout p_{12} with $Q^{0\text{-normvert}}(p_{12}) \approx 0.45$, $Q^{0\text{-normendv}}(p_{12}) \approx 1.22$

Layout p_{13} with $Q^{-2\text{-normvert}}(p_{13}) \approx 0.67$, $Q^{-2\text{-normendv}}(p_{13}) \approx 0.95$

Layouts with approximately optimal r-normalized$^{\text{vert}}$ atedge length



Layout p_{21} with $Q^{1\text{-normvert}}(p_{21}) \approx 0.46$, $Q^{1\text{-normendv}}(p_{21}) \approx 0.43$

Layout p_{22} with $Q^{0\text{-normvert}}(p_{22}) \approx 0.69$, $Q^{0\text{-normendv}}(p_{22}) \approx 0.71$

Layout p_{23} with $Q^{-2\text{-normvert}}(p_{23}) \approx 0.77$, $Q^{-2\text{-normendv}}(p_{23}) \approx 0.85$

Layouts with approximately optimal r-normalized$^{\text{endv}}$ atedge length

Figure 6.3: Layouts of a graph without vertex weights. The edges between the six central vertices have weight 2, the other edges have weight 1.

Equivalence to the Normalized Atedge Length The following theorem states that for a given connected graph and $r < 1$, a layout with minimum r-LinPoly energy also has minimum r-normalized atedge length. The restriction to connected graphs is no serious loss of generality, because a separate layout can be computed for each connected component of a graph. The restriction to $r < 1$ excludes only the interesting case of $r = 1$, because the r-scaled and r-normalized atedge length were only motivated for $r \leq 1$ in Subsection 6.1.2. Minima of the 1-LinPoly energy indeed are not non-trivial minima of the r-normalized atedge length, but usually either layouts where all vertices have the same position, or layouts where distances approach infinity. Anyway, $r < 1$ is more useful for many practical applications.

Theorem 6.2 *Let $G = (\mathcal{V}, \mathcal{E})$ be a connected graph, and let $r \in \mathbb{R}$ with $r < 1$. Then G has a layout with minimum r-LinPoly energy, and this layout has minimum r-normalized atedge length.*

Proof: If the distance between two vertices approaches infinity, then the length of at least one edge approaches infinity because G is connected, and thus the r-LinPoly

energy approaches (positive) infinity (for $r<1$). Thus the distances between vertices in layouts with minimum r-LinPoly energy are finite, and there are layouts with minimum r-LinPoly energy and finite coordinates.

Let p_0 be a layout of G with minimum r-LinPoly energy. It remains to show that p_0 also minimizes the r-normalized atedge length. The basic idea is to fix the total atedge length (i.e. the first term of the r-LinPoly energy) temporarily. This does not restrict generality, but only the scaling factor, and thus can be removed at the end of the proof. It allows to transform the minimization of energy into a minimization of the inverse power mean of the atvertex distances.

The following proof is for $r \neq 0$, the proof for $r = 0$ is similar. Let the layout p_0 be a solution of the minimization problem:

$$\text{Minimize } \mathrm{U}^r(\mathrm{p}).$$

Let $c := \mathrm{Q}(\mathrm{p}_0)$. Then p_0 is also a solution of

$$\text{Minimize } \mathrm{U}^r(\mathrm{p}) \text{ subject to } \mathrm{Q}(\mathrm{p}) = c.$$

This is equivalent to

$$\text{Minimize } u_r(\mathrm{p}) \text{ subject to } \mathrm{Q}(\mathrm{p}) = c,$$

where $u_r(\mathrm{p}) := -\sum_{\{v_1,v_2\}\in\mathcal{V}^{\langle 2\rangle}} \frac{1}{r}\|\mathrm{p}(v_1)-\mathrm{p}(v_2)\|^r$. For all layouts p holds $u_r(\mathrm{p}) \geq 0$ if $r < 0$, and $u_r(\mathrm{p}) \leq 0$ if $r > 0$. Thus $(-r u_r(\mathrm{p}))^{-1/r}$ is monotonically increasing with $u_r(\mathrm{p})$, and p_0 is a solution of

$$\text{Minimize } (-r u_r(\mathrm{p}))^{-1/r} \text{ subject to } \mathrm{Q}(\mathrm{p}) = c.$$

Because c is nonnegative, p_0 is also a solution of

$$\text{Minimize } \mathrm{Q}^{r\text{-scal}}(\mathrm{p}) = \frac{\mathrm{Q}(\mathrm{p})}{(-r u_r(\mathrm{p}))^{1/r}} \text{ subject to } \mathrm{Q}(\mathrm{p}) = c.$$

The condition $\mathrm{Q}(\mathrm{p}) = c$ can be removed, because for any layout p_1 that minimizes $\mathrm{Q}^{r\text{-scal}}(\mathrm{p})$, there exists a layout p_2 with $\mathrm{Q}^{r\text{-scal}}(\mathrm{p}_1) = \mathrm{Q}^{r\text{-scal}}(\mathrm{p}_2)$ and $\mathrm{Q}(\mathrm{p}_2) = c$: If $c = 0$, then $\mathrm{Q}^{r\text{-scal}}(\mathrm{p}_0) = 0$, thus $\mathrm{Q}^{r\text{-scal}}(\mathrm{p}_1) = 0$ (because p_1 is a minimum), and thus $\mathrm{Q}(\mathrm{p}_1) = 0 = c$. If $c > 0$, then $\mathrm{Q}^{r\text{-scal}}(\mathrm{p}_0) > 0$, thus $\mathrm{Q}^{r\text{-scal}}(\mathrm{p}_1) > 0$ (because p_0 is a minimum), thus $\mathrm{Q}(\mathrm{p}_1) > 0$, and thus scaling p_1 with $\frac{c}{\mathrm{Q}(\mathrm{p}_1)}$ (i.e. multiplying all positions of p_1 with this factor) yields a layout p_2 with $\mathrm{Q}^{r\text{-scal}}(\mathrm{p}_1) = \mathrm{Q}^{r\text{-scal}}(\mathrm{p}_2)$ and $\mathrm{Q}(\mathrm{p}_2) = c$. So p_0 is also a solution of

$$\text{Minimize } \mathrm{Q}^{r\text{-scal}}(\mathrm{p}).$$

which is (for a fixed graph) equivalent to

$$\text{Minimize } \mathrm{Q}^{r\text{-norm}}(\mathrm{p}).$$

□

All layouts with minimum r-LinPoly energy also have minimum r-normalized atedge length, according to Theorem 6.2. But not vice versa, because scaling a layout (i.e. multiplying all coordinates with a constant factor) affects the r-LinPoly energy, but not the r-normalized atedge length. The following theorem gives the scaling of layouts with minimum r-LinPoly energy. Besides demonstrating a nice proof technique, it has also practical applications e.g. in determining an appropriate scaling for initial layouts of iterative energy minimization.

Theorem 6.3 *Let $G = (\mathcal{V}, \mathcal{E})$ be a graph, let $r \in \mathbb{R}$ with $r < 1$, and let p_0 be a layout of G with minimum r-LinPoly energy. Then*

$$\sum_{\{v_1,v_2\}\in\mathcal{E}} \|\mathrm{p}_0(v_1)-\mathrm{p}_0(v_2)\| \;=\; \sum_{\{v_1,v_2\}\in\mathcal{V}^{\langle 2\rangle}} \|\mathrm{p}_0(v_1)-\mathrm{p}_0(v_2)\|^r \;.$$

Proof: The basic idea of the proof is to scale the minimum energy layout p_0, and to express the r-LinPoly energy of the resulting layouts as a function of a scaling factor. This function has a minimum at the minimum energy layout p_0, i.e. at a scaling factor of 1.

The following proof is for $r \neq 0$, the proof for $r = 0$ is similar. The r-LinPoly energy of the layout p_0 is

$$\sum_{\{v_1,v_2\}\in\mathcal{E}} \|\mathrm{p}_0(v_1)-\mathrm{p}_0(v_2)\| \;-\; \sum_{\{v_1,v_2\}\in\mathcal{V}^{\langle 2\rangle}} \frac{1}{r}\|\mathrm{p}_0(v_1)-\mathrm{p}_0(v_2)\|^r \;.$$

If all coordinates in p_0 are multiplied with some $d \in \mathbb{R}_+$, the r-LinPoly energy is

$$u(d) \;:=\; \sum_{\{v_1,v_2\}\in\mathcal{E}} d\|\mathrm{p}_0(v_1)-\mathrm{p}_0(v_2)\| \;-\; \sum_{\{v_1,v_2\}\in\mathcal{V}^{\langle 2\rangle}} \frac{1}{r}d^r\|\mathrm{p}_0(v_1)-\mathrm{p}_0(v_2)\|^r \;.$$

Because p_0 is a layout with minimal r-LinPoly energy, this function has a global minimum at $d = 1$, so $u'(1) = 0$.

$$\begin{aligned} u'(d) &= \sum_{\{v_1,v_2\}\in\mathcal{E}} \|\mathrm{p}_0(v_1)-\mathrm{p}_0(v_2)\| \;-\; d^{r-1}\sum_{\{v_1,v_2\}\in\mathcal{V}^{\langle 2\rangle}} \|\mathrm{p}_0(v_1)-\mathrm{p}_0(v_2)\|^r \\ 0 = u'(1) &= \sum_{\{v_1,v_2\}\in\mathcal{E}} \|\mathrm{p}_0(v_1)-\mathrm{p}_0(v_2)\| \;-\; \sum_{\{v_1,v_2\}\in\mathcal{V}^{\langle 2\rangle}} \|\mathrm{p}_0(v_1)-\mathrm{p}_0(v_2)\|^r \end{aligned}$$

□

6.3 Distance vs. Inter-Density of Subgraphs

The parameter r of the r-normalized atedge length and the r-LinPoly energy model was introduced to control how closely dense subgraphs are grouped, and how distantly sparsely connected subgraphs are placed. For $r \approx 0$, the distances of subgraphs in layouts with minimum r-LinPoly energy reflect their inter-density, while for $r \to -\infty$, the distances of subgraphs are basically uniform and thus independent of their inter-density.

This is formalized, for the simple case of graphs with two vertices, in the following Theorem 6.4. It states that the distance between the two vertices in a layout with minimum r-LinPoly energy (for $r < 1$) is the $\frac{1}{r-1}$th power of their inter-density. This means, for example, that the distance is the inverse of the inter-density in layouts with minimum LinLog energy (i.e. if $r = 0$), and that the distance almost is independent of the inter-density if $r \to -\infty$.

The case of two vertices may appear to be too trivial to be interesting, but it demonstrates a proof method that can be extended to more complex graphs and layouts (see [Noa04]), and it approximates more general and practically relevant situations. For example, let p be a clustering of a graph G with two dense, sparsely connected clusters c_1 and c_2. Let p be a layout of G with minimum r-LinPoly energy. Due to the large intra-density and small inter-density of the clusters, the distances of the vertices *within* c_1 and *within* c_2 in p are much smaller than the distances *between* c_1 and c_2 (unless r is very small). This can be approximated by assuming that all vertices of c_1 have the same position and all vertices of c_2 have the same position, and thus by considering c_1 and c_2 as two big vertices.

Theorem 6.4 *Let $G = ((V, \mathrm{w}), (E, \mathrm{f}))$ be a graph with two vertices v_1, v_2 and one edge $\{v_1, v_2\}$, let $r \in \mathbb{R}$ with $r < 1$, and let p_0 be a layout of G with minimum r-LinPoly energy. Then $\|\mathrm{p}_0(v_1) - \mathrm{p}_0(v_2)\| = \mathrm{den}(v_1, v_2)^{1/(r-1)}$.*

Proof: The following proof is for $r \neq 0$, the proof for $r = 0$ is similar. Let p be a layout of G, and let $d := \|\mathrm{p}(v_1) - \mathrm{p}(v_2)\|$. Then the r-LinPoly energy of p is

$$u(d) \;:=\; \mathrm{f}(\{v_1, v_2\})d - \mathrm{w}(v_1)\mathrm{w}(v_2)\frac{1}{r}d^r \;.$$

Let $d_0 := \|\mathrm{p}_0(v_1) - \mathrm{p}_0(v_2)\|$. Because p_0 is a layout with minimum energy, the function $u(d)$ has a global minimum at d_0, so $u'(d_0) = 0$.

$$\begin{aligned} 0 \;=\; u'(d_0) \;&=\; \mathrm{f}(\{v_1, v_2\}) - \mathrm{w}(v_1)\mathrm{w}(v_2)d_0^{r-1} \\ d_0 \;&=\; \left(\frac{\mathrm{f}(\{v_1, v_2\})}{\mathrm{w}(v_1)\mathrm{w}(v_2)}\right)^{1/(r-1)} \;=\; \mathrm{den}(v_1, v_2)^{1/(r-1)} \end{aligned}$$

□

6.4 Related Work

Techniques for computing layouts are known as multidimensional scaling, dimensionality reduction, and graph drawing. *Multidimensional scaling* takes as input pairwise dissimilarities of a set of objects, and maps each object to a position in low-dimensional space such that the distances of the positions match the (probably transformed) dissimilarities of the objects [BG97, CC01]. *Dimensionality reduction* maps vectors in a high-dimensional space to vectors in a low-dimensional space, such that the distances between the vectors are approximately preserved (sometimes globally, sometimes only locally). Recent overviews can be found in articles by Xu and

Wunsch [XW05, Section II.L] and Saul and Roweis [SR03, Section 7], more comprehensive treatments are available for individual dimensionality reduction techniques like principal component analysis [Jol02], independent component analysis [HKO01], and self-organizing maps [Koh01]. *Graph drawing* maps each vertex of a graph to a point and each edge to a curve in a low-dimensional space [DETT99, KW01]. When the edges are represented as straight lines or not at all, graph drawing reduces to finding positions for the vertices, i.e. to constructing a graph layout as defined in Section 2.3. *Energy-based graph drawing methods* comprise explicit quality measures for graph layouts called energy models, and algorithms for computing layouts with small energy. Similarly, *force-directed graph drawing methods* specify forces acting on each vertex, and an algorithm searches for an equilibrium state where the total force on each vertex is zero. Because force is the negative gradient of energy, equilibria of forces are local minima of energy.

This section focuses on quality measures for graph layouts. This excludes multidimensional scaling and dimensionality reduction, which construct layouts for dissimilarity data and vectorial data, respectively. Multidimensional scaling has been applied to compute graph layouts, using the graph-theoretic distance (i.e. the length of the shortest path) between vertices as input dissimilarity (e.g. [KS80, KK89, Coh97, BP07, CMIBR07]), but this approach presumes that the edge weights of the graph have dissimilarity semantics. A recent paper of Hachul and Jünger [HJ06] provides an overview of efficient algorithms for energy minimization, including an experimental comparison of some of these algorithms.

The first subsection introduces a uniform template that is used in the following subsections to describe the layout quality measures. The last subsection discusses related work on the analysis and validation of layout quality measures.

6.4.1 Description Template

The formal definitions in this section are given for a layout p of a graph without vertex weights $(V, \mathcal{E})$, because most related literature does not consider vertex weights. All layout quality measures are described using the following template:

Definition: Definition of the layout quality measure as distance ratio, i.e. in a form similar to the r-normalized atedge length. Some quality measures were originally proposed as force model. In this case, the force model is first transformed into an energy model (i.e. into a form similar to the r-LinPoly energy model) by exploiting that force is the negative gradient of energy. Then this energy model is transformed into a distance ratio as in the proof of Theorem 6.2.

References: The original definition of the measure and a reference to its source. If the original definition contains parameters that only affect the scaling of the layout, these parameters are omitted for simplicity.

Bias: Because all quality measures are presented uniformly as distance ratios, the Theorems 6.1, 3.2, and 3.3 can be easily applied or adapted to identify their

biases. The desirability of some biases strongly depends on the application scenario. This holds in particular for strong biases towards uniform vertex distances or uniform edge lengths, which may improve the readability of box-line visualizations, but impair the interpretability with respect to the density or sparsity of subgraphs.

Vertex Distance: For an optimal layout of a graph with two vertices v_1 and v_2, the dependency of the distance between v_1 and v_2 on their inter-density $\mathrm{den}^{\mathrm{vert}}(v_1, v_2)$ or $\mathrm{den}^{\mathrm{endv}}(v_1, v_2)$. This dependency is derived as in the proof of Theorem 6.4. If the distance is strongly dependent on the inter-density (e.g. inversely proportional), then subgraphs with small inter-density are separated in optimal layouts, and the inter-density is reflected by their distance. Otherwise (e.g. if the distance is $\mathrm{den}^{\mathrm{vert}}(v_1, v_2)^{-1/3}$), the optimal layouts separate such subgraphs only if the inter-density is *much* smaller than the intra-density. Many existing force and energy models belong to the second category, because they enforce uniform vertex distances or edge lengths to improve the readability of box-line visualizations.

6.4.2 Normalized$^{\mathrm{vert}}$ Measures

LinLog$^{\mathrm{vert}}$

Definition: $\dfrac{\sum_{\{v_1,v_2\}\in\mathcal{E}} \|\mathrm{p}(v_1)-\mathrm{p}(v_2)\|}{|\mathcal{E}|} \;/\; \left(\prod_{\{v_1,v_2\}\in V^{(2)}} \|\mathrm{p}(v_1)-\mathrm{p}(v_2)\|\right)^{1/|V^{(2)}|}$

i.e. the quotient of the arithmetic mean of the atedge lengths and the geometric mean of the vertex distances.

References: Introduced as LinLog energy model by the present author in [Noa04]:

$$\sum_{\{v_1,v_2\}\in\mathcal{E}} \|\mathrm{p}(v_1)-\mathrm{p}(v_2)\| \;-\; \sum_{\{v_1,v_2\}\in V^{(2)}} \ln \|\mathrm{p}(v_1)-\mathrm{p}(v_2)\|$$

Bias:
- Bias towards uniform vertex distances (Theorem 6.1$^{\mathrm{vert}}$); the energy model is additionally biased towards a certain scaling (Theorem 6.3$^{\mathrm{vert}}$).
- Bias towards high-degree vertices on central positions (Theorem 3.2).

Vertex Distance: $\mathrm{den}^{\mathrm{vert}}(v_1, v_2)^{-1}$

Fruchterman-Reingold

Definition: $\left(\dfrac{\sum_{\{v_1,v_2\}\in\mathcal{E}} \|\mathrm{p}(v_1)-\mathrm{p}(v_2)\|^3}{|\mathcal{E}|}\right)^{1/3} \;/\; \left(\prod_{\{v_1,v_2\}\in V^{(2)}} \|\mathrm{p}(v_1)-\mathrm{p}(v_2)\|\right)^{1/|V^{(2)}|}$

i.e. the quotient of the third power mean of the atedge lengths and the geometric mean of the vertex distances.

References: Introduced as force model by Fruchterman and Reingold [FR91], with the following force acting on each vertex $v_1 \in V$:

$$\sum_{\{v_1,v_2\}\in\mathcal{E}} \|\mathrm{p}(v_1)-\mathrm{p}(v_2)\|^2 \cdot \overrightarrow{\mathrm{p}(v_1)\mathrm{p}(v_2)} \;+\; \sum_{\{v_1,v_2\}\in V^{(2)}} \|\mathrm{p}(v_1)-\mathrm{p}(v_2)\|^{-1} \cdot \overrightarrow{\mathrm{p}(v_2)\mathrm{p}(v_1)}$$

where $\overrightarrow{\mathrm{p}(v_1)\mathrm{p}(v_2)}$ is the unit length vector $\frac{\mathrm{p}(v_2)-\mathrm{p}(v_1)}{\|\mathrm{p}(v_2)-\mathrm{p}(v_1)\|}$ pointing from v_1 to v_2.

The Fruchterman-Reingold model has been used, sometimes with minor modifications, in many subsequent works on force-directed graph drawing.

Bias:
- Strong bias towards uniform edge lengths, bias towards uniform vertex distances (Theorem 6.1^{vert}); the force model is additionally biased towards a certain scaling (Theorem 6.3^{vert}).
- Bias towards high-degree vertices on central positions (Theorem 3.2).

Vertex Distance: $\mathrm{den}^{\mathrm{vert}}(v_1,v_2)^{-1/3}$

Related: The seminal force model in graph drawing was introduced by Eades [Ead84]. It has the same form as the Fruchterman-Reingold model, except that the repulsive force acts only between nonadjacent vertices:

$$\sum_{\{v_1,v_2\}\in\mathcal{E}} \log\|\mathrm{p}(v_1)-\mathrm{p}(v_2)\|\cdot\overrightarrow{\mathrm{p}(v_1)\mathrm{p}(v_2)} + \sum_{\{v_1,v_2\}\in V^{(2)}\setminus E} \|\mathrm{p}(v_1)-\mathrm{p}(v_2)\|^{-2}\cdot\overrightarrow{\mathrm{p}(v_2)\mathrm{p}(v_1)}\,.$$

Even if the repulsive force is extended to all pairs of vertices, the dependency between distance and inter-density is still weak (between $\mathrm{den}^{\mathrm{vert}}(v_1,v_2)^{-1/2}$ and $\mathrm{den}^{\mathrm{vert}}(v_1,v_2)^{-1/3}$ for $\mathrm{den}^{\mathrm{vert}}(v_1,v_2) \ll 1$).

Davidson-Harel

Definition: $$\left(\frac{\sum_{\{v_1,v_2\}\in\mathcal{E}} \|\mathrm{p}(v_1)-\mathrm{p}(v_2)\|^2}{|\mathcal{E}|}\right)^{1/2} \Big/ \left(\frac{\sum_{\{v_1,v_2\}\in V^{(2)}} \|\mathrm{p}(v_1)-\mathrm{p}(v_2)\|^{-2}}{|V^{(2)}|}\right)^{-1/2}$$
i.e. the quotient of the second power mean of the atedge lengths and the minus second power mean of the vertex distances.

References: Introduced as energy model by Davidson and Harel [DH96]:

$$\textstyle\sum_{\{v_1,v_2\}\in\mathcal{E}} \|\mathrm{p}(v_1)-\mathrm{p}(v_2)\|^2 \;+\; \sum_{\{v_1,v_2\}\in V^{(2)}} \|\mathrm{p}(v_1)-\mathrm{p}(v_2)\|^{-2}$$

Davidson and Harel proposed further types of forces, see Subsection 6.4.4.

Bias:
- Bias towards uniform edge lengths and uniform vertex distances (Theorem 6.1^{vert}); the energy model is additionally biased towards a certain scaling (Theorem 6.3^{vert}).
- Bias towards high-degree vertices on central positions (Theorem 3.2).

Vertex Distance: $\mathrm{den}^{\mathrm{vert}}(v_1,v_2)^{-1/4}$

Hall-Fiedler

Definition: $\left(\frac{\sum_{\{v_1,v_2\}\in\mathcal{E}} \|\mathrm{p}(v_1)-\mathrm{p}(v_2)\|^2}{|\mathcal{E}|}\right) \Big/ \left(\frac{\sum_{\{v_1,v_2\}\in V^{(2)}} \|\mathrm{p}(v_1)-\mathrm{p}(v_2)\|^2}{|V^{(2)}|}\right)$

i.e. the quotient of the mean squared atedge length and the mean squared vertex distance. The square root of this measure (which has the same minima for a fixed graph) is the quotient of the second power mean of the atedge lengths and the second power mean of the vertex distances.

References: Introduced by Hall [Hal70] for one-dimensional layouts p as $\sum_{\{v_1,v_2\}\in\mathcal{E}} \|\mathrm{p}(v_1)-\mathrm{p}(v_2)\|^2$, with the constraint $\sum_{v\in V} \mathrm{p}(v)^2 = 1$ to avoid that all vertices are assigned to the same position. Fiedler observed in a different context that solutions of this minimization problem are also minima of the above quotient [Fie75], and Koren first applied this observation to graph layouts [Kor05].

The quality measure is minimized by the eigenvector corresponding to the second smallest eigenvalue of the Laplacian of the graph (see Subsection 3.2.2). Hall states explicitly that he chose the measure because of its mathematical tractability [Hal70, Section 7].

With the Hall-Fiedler quality measure, an optimal one-dimensional layout is also an optimal layout in any (positive) number of dimensions. In practice, non-trivial higher-dimensional layouts are constructed by combining several one-dimensional layouts that minimize the quality measure subject to mutual orthogonality [Hal70, Kor05]. (Two one-dimensional layouts p_1 and p_2 are called orthogonal if $\sum_{v\in V} \mathrm{p}_1(v)\mathrm{p}_2(v) = 0$.)

After Hall's original proposal in 1970, spectral graph drawing has not been used much for decades, but has been revitalized mainly by Koren [KCH03, Kor05] and Brandes et al. (e.g. [BC01a, BW02]).

Bias:
- No bias towards any layout for graphs with uniform density$^{\mathrm{vert}}$ (Theorem 6.1$^{\mathrm{vert}}$).
- Bias towards high-degree vertices on central positions (Theorem 3.2).

Vertex Distance: Not applicable, because there is no equivalent energy model of a form similar to r-LinPoly.

Discussion: Global minima of the Hall-Fiedler quality measure can be efficiently computed, but the resulting layouts have serious deficiencies. First, no layout (in any number of dimensions) can be better than the best one-dimensional layout. Second, many vertices may be placed at the same position (see e.g. [HJ06, Kor05] for empirical evidence). This problem may not occur for the other quality measures in this subsection, because they approach infinity when the distance of any two vertices approaches 0. It motivated the generalization of the scaled atedge length to the r-scaled atedge length in Subsection 6.1.2.

6.4.3 Normalized$^{\text{endv}}$ Measures

LinLog$^{\text{endv}}$

Definition: $\dfrac{\sum_{\{v_1,v_2\}\in\mathcal{E}} \|\mathrm{p}(v_1)-\mathrm{p}(v_2)\|}{|\mathcal{E}|} \Big/ \left(\prod_{\{v_1,v_2\}\in(V,\deg)^{\langle 2\rangle}} \|\mathrm{p}(v_1)-\mathrm{p}(v_2)\|\right)^{1/|(V,\deg)^{\langle 2\rangle}|}$

i.e. the quotient of the arithmetic mean of the atedge lengths and the geometric mean of the endvertex distances.

References: Introduced as edge-repulsion LinLog energy model by the present author in [Noa06]:

$$\sum_{\{v_1,v_2\}\in\mathcal{E}} \|\mathrm{p}(v_1)-\mathrm{p}(v_2)\| \;-\; \sum_{\{v_1,v_2\}\in(V,\deg)^{\langle 2\rangle}} \ln\|\mathrm{p}(v_1)-\mathrm{p}(v_2)\|$$

Bias:
- Bias towards uniform endvertex distances (Theorem 6.1$^{\text{endv}}$); the energy model is additionally biased towards a certain scaling (Theorem 6.3$^{\text{endv}}$).

Vertex Distance: $\mathrm{den}^{\text{endv}}(v_1,v_2)^{-1}$

Discussion: The second term of the energy model can be interpreted as repulsion between atedges, with the restriction that the repulsion does not act between entire atedges, but only between their endvertices. So the energy model is symmetric in the sense that atedges cause both attraction and repulsion. In other words, vertices that attract strongly (through many incident atedges) also repulse strongly.

Chung-Koren

Definition: $\left(\dfrac{\sum_{\{v_1,v_2\}\in\mathcal{E}} \|\mathrm{p}(v_1)-\mathrm{p}(v_2)\|^2}{|\mathcal{E}|}\right) \Big/ \left(\dfrac{\sum_{\{v_1,v_2\}\in(V,\deg)^{\langle 2\rangle}} \|\mathrm{p}(v_1)-\mathrm{p}(v_2)\|^2}{|(V,\deg)^{\langle 2\rangle}|}\right)$

i.e. the quotient of the mean squared atedge length and the mean squared endvertex distance.

References: Occurred without reference to graph layouts (as a characterization of a generalized eigenvalue of the Laplacian) in a book on spectral graph theory by Chung [Chu97, Section 1.2]. Introduced to the field of graph drawing by Koren [Kor03, Kor05]. The quality measure is minimized by a generalized eigenvector of the Laplacian of the graph (see Subsection 3.2.2).

See the Hall-Fiedler quality measure for remarks on higher-dimensional layouts and a discussion.

Bias:
- No bias towards any layout for graphs with uniform density$^{\text{endv}}$ (Theorem 6.1$^{\text{endv}}$).

6.4.4 Extensions

Unconnected Graphs If a graph has more than one connected component, the distances of the connected components approach infinity in the optimal layouts of most quality measures. The connectedness of a graph can be insured by adding a vertex of weight 0, and connecting it to all other vertices with edges of small weight. Alternatively, a special gravitational force or energy can be introduced that attracts each vertex to the barycenter of the layout [FLM95]. Of course, the distortions caused by additional edges or forces impair the interpretability of the resulting layouts, so it is advisable to keep them as small as possible, or to avoid them entirely by analyzing each connected component separately.

Graphs with Clusterings The techniques of the previous paragraph can be generalized to group the vertices according to a *given* graph clustering: For each cluster, a virtual vertex of weight 0 is introduced and connected to each vertex of the cluster [EH00, NL05].

Extensions for Improved Readability Many other types of forces and energies have been proposed besides attraction between adjacent vertices and repulsion between different vertices. Such proposals include repulsion between vertices and edges [DH96, CP96], penalties for edge crossings [DH96, CP96], forces for widening the angle between neighboring edges of a vertex [MRS96, LY05], and repulsion from the border of the drawing area [DH96]. All these forces are intended to improve the readability of layouts, and not to enable interpretations concerning the density of subgraphs.

6.4.5 Validation of Layout Quality Measures

Analytical Validation As detailed in the paragraphs on the Hall-Fiedler and the Chung-Koren quality measure, several researchers have characterized spectral layouts as minima of certain distance ratios or constrained optimization problems. However, Theorem 6.2 is the first such characterization for force-directed or energy-based layouts in graph drawing.

Previous works have also established various connections between layouts and clusterings. In particular, layouts and clusterings have been used as complementary models of dissimilarity data (see Subsection 3.2.1), and layouts are intermediate results in spectral algorithms (see Subsection 3.2.2) and some flow-based algorithms (e.g. [AR98, LLR95]) for computing graph clusters. However, Theorem 6.4 is the first result that relates the inter-density of subgraphs to their distance in layouts.

Empirical Validation Existing empirical studies of force-directed and energy-based graph drawing methods by Brandenburg et al. [BHR96b] and Hachul and Jünger [HJ06] focus on the readability of box-line visualizations, while the goal of this work is the faithful representation of density and sparsity.

6.5 Examples

Like Section 5.3 for orderings, this section serves three purposes. First, it illustrates the Fruchterman-Reingold force model (see Section 6.4), the 0-normalized$^{\text{vert}}$ atedge length, and the 0-normalized$^{\text{endv}}$ atedge length, with their different biases, for layouts of real-world graphs.[1] Second, it provides empirical evidence for the internal and external validity of the 0-normalized$^{\text{vert}}$ atedge length and the 0-normalized$^{\text{endv}}$ atedge length as layout quality measures. Third, it empirically compares these new measures with the well-known Fruchterman-Reingold force model, whose results are similar to those of the other popular force and energy models by Eades and Davidson-Harel.

Method All graphs in this section are described in Appendix A.1. For each graph, a layout with small Fruchterman-Reingold energy, a layout with small LinLog$^{\text{vert}}$ energy (and thus small 0-normalized$^{\text{vert}}$ atedge length), and a layout with small LinLog$^{\text{endv}}$ energy (and thus small 0-normalized$^{\text{endv}}$ atedge length) are presented in box-line visualizations. The layouts were computed with the force calculation heuristic of Barnes and Hut [BH86]. This heuristic provides no guarantee that the computed local energy minima are globally optimal, and global optimality is not assumed in the following. Layouts were rotated manually to a enable a space-efficient placement of the figures, and to facilitate the comparison of the different layouts for each graph; rotation does not change the layout quality.

Validation methods for graph assignments and assignment quality measures were introduced in Section 1.3. Internal validity of a layout quality measure requires that in layouts with small measurement values, densely connected vertices have small distances, and sparsely connected vertices have large distances. Because the edges are elided in all box-line visualizations to avoid clutter, this can only be checked using the matrix visualizations in Section 5.3. External validity of a layout quality measure requires that layouts with small measurement values are externally valid, i.e. resemble known authoritative groupings. This is checked by comparing the computed layouts with the authoritative groupings described in Appendix A.1.

Results The layouts are shown in Figures 6.4 to 6.11 on pages 114 to 121.

In the Fruchterman-Reingold layouts of World Trade and US Airlines (as graphs with very nonuniform degrees), the vertex positions mainly reflect the vertex degrees. In the other Fruchterman-Reingold layouts, the vertices are distributed fairly uniformly, and no clear separation of vertex groups is apparent. The remainder of this paragraph focuses on the layouts with small 0-normalized$^{\text{vert}}$ atedge length and small 0-normalized$^{\text{endv}}$ atedge length.

Concerning internal validity, the layouts with small 0-normalized$^{\text{endv}}$ atedge length indeed group densely$^{\text{endv}}$ connected vertices and separate sparsely$^{\text{endv}}$ connected vertices for all examined graphs, although the evaluation is somewhat difficult

[1] The case $r = 0$ of the r-normalized atedge length is used because in its optimal layouts, the distance of subgraphs is *proportional* to their inter-density (see Section 6.3).

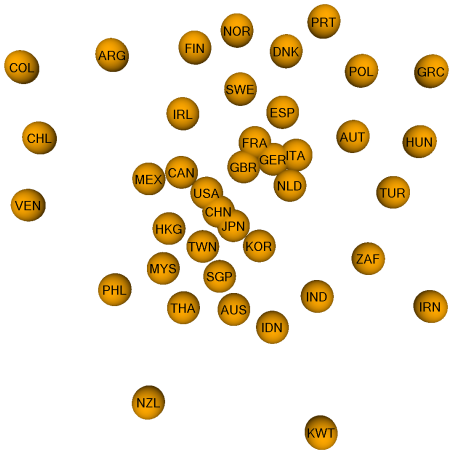
due to the need for referring to the matrix visualizations. The observations for the orderings with small 0-normalized$^{\text{vert}}$ atedge length are similar.

Concerning external validity, the layouts with small 0-normalized$^{\text{endv}}$ atedge length conform well to the authoritative groupings. The layout of World Trade groups the countries of each main economical area (East Asia / Australia, America, and Europe), but also other closely interlocked countries like HKG and CHN, AUS and NZL, GBR and IRL, ESP and PRT, and North Europe. The layout of US Airlines resembles the relative geographical distances of the airports remarkably closely, given that the graph does not contain any explicit information about geographical distances. The layout of ODLIS groups semantically related terms well; because of the size of the graph, static visualizations can show this only to a limited extent. The layout of Southern Women not only reflects the clusters of Freeman's consensus study, but also shows that some women (particularly Pearl, but also Ruth, Dorothy, and Verne) are rather between the two groups than clearly within one group. The layout of Karate Club clearly separates the authoritative clusters. In the layout of Morse Code Confusion, the horizontal positions are only loosely related to the number of beeps, and the vertical positions are only loosely related to the fraction of short beeps in the signal. The layout of Food Classification reflects not only the major groups of the authoritative decomposition, but also many additional details, like the similarity of the grain rice to vegetables, and the tendency of spaghetti towards meats. The layout of College Football clearly groups the teams of each conference.

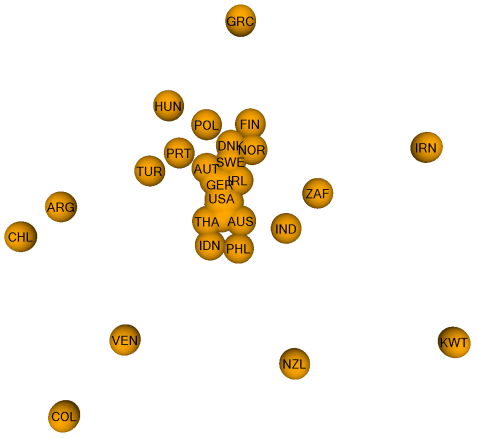
The layouts with small 0-normalized$^{\text{vert}}$ atedge length mainly group the vertices according to their degree for World Trade, US Airlines, and ODLIS (the graphs with the most nonuniform degrees), and thus do not conform to the authoritative groupings for these graphs. The layout for Southern Women mainly separates Flora and Olivia from the remaining women, which deviates from Freeman's consensus clustering, but is reasonable because these women take part in very few events. The layout for Karate Club shows a similar tendency to separate vertices with small degrees, but still reflects the authoritative clustering. The layout for Morse Code Confusion lines up signals with the same number of beeps, ordered by their fraction of short beeps. The groups in the layout for Food Classification are similar to the layout with small 0-normalized$^{\text{endv}}$ atedge length[2], and the layout for College Football is very similar to the layout with small 0-normalized$^{\text{endv}}$ atedge length.

In summary, the examined layouts with small 0-normalized$^{\text{endv}}$ atedge length conform very well to the authoritative groupings. The only exception is Morse Code Confusion, where the layout with small 0-normalized$^{\text{vert}}$ atedge length is arguably somewhat better. The examined layouts with small Fruchterman-Reingold energy do not show the authoritative groupings, but either no clear groups at all (for graphs with fairly uniform degrees), or groups that reflect mainly vertex degrees.

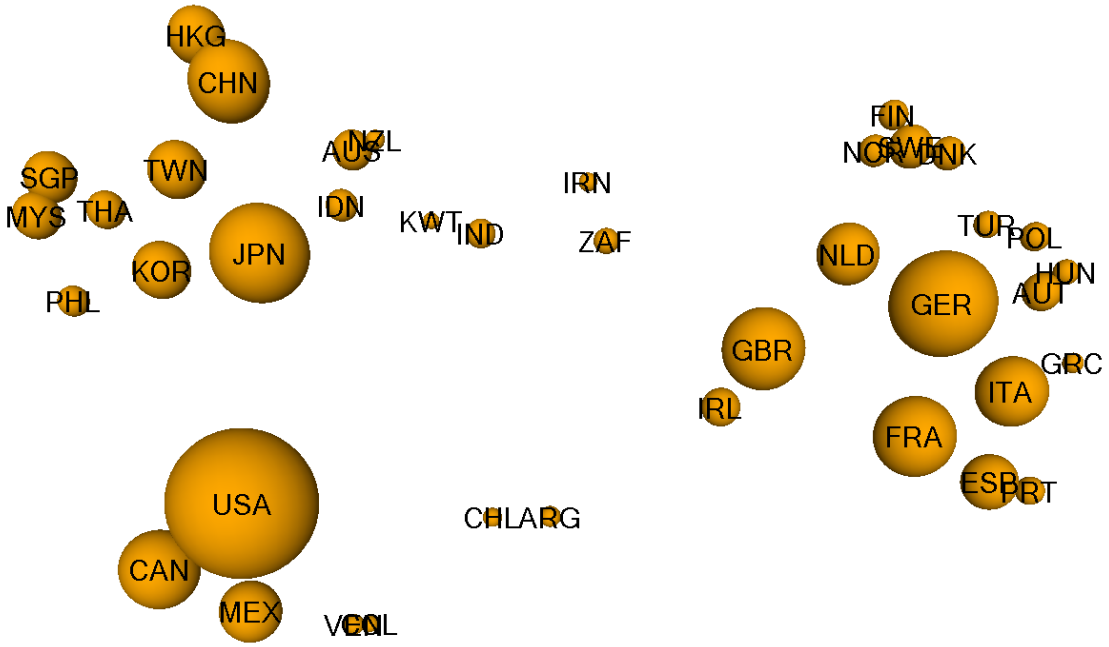
[2]The Food Classification graph has fairly uniform degrees, and thus the 0-normalized$^{\text{vert}}$ atedge length is very close to the 0-normalized$^{\text{endv}}$ atedge length for every layout of this graph. Nevertheless, the optimal layouts for the two measures differ significantly (Figure 6.10b and 6.10c). The reason is that both layouts are near-optimal for both measures, but one layout is slightly better for one measure, and the other layout is slightly better for the other measure.



(a) Layout with small Fruchterman-Reingold energy

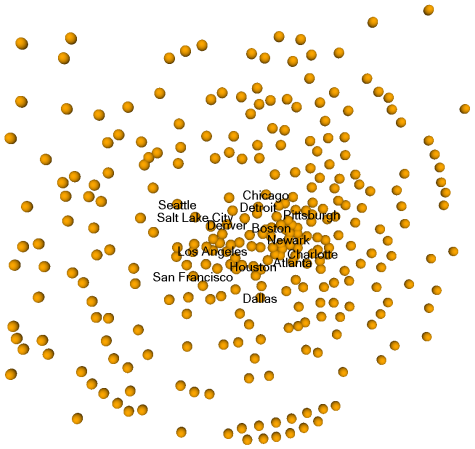


(b) Layout with small LinLog^{vert} energy; normalized^{vert} atedge length 0.226

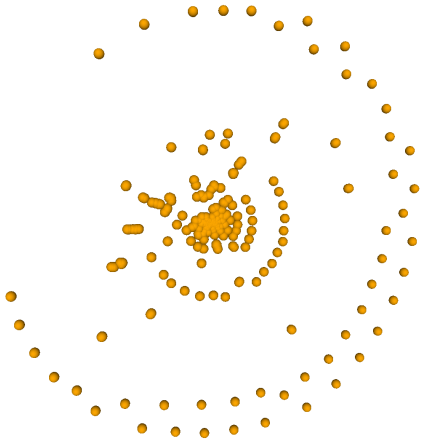


(c) Layout with small LinLog^{endv} energy; normalized^{endv} atedge length 0.598

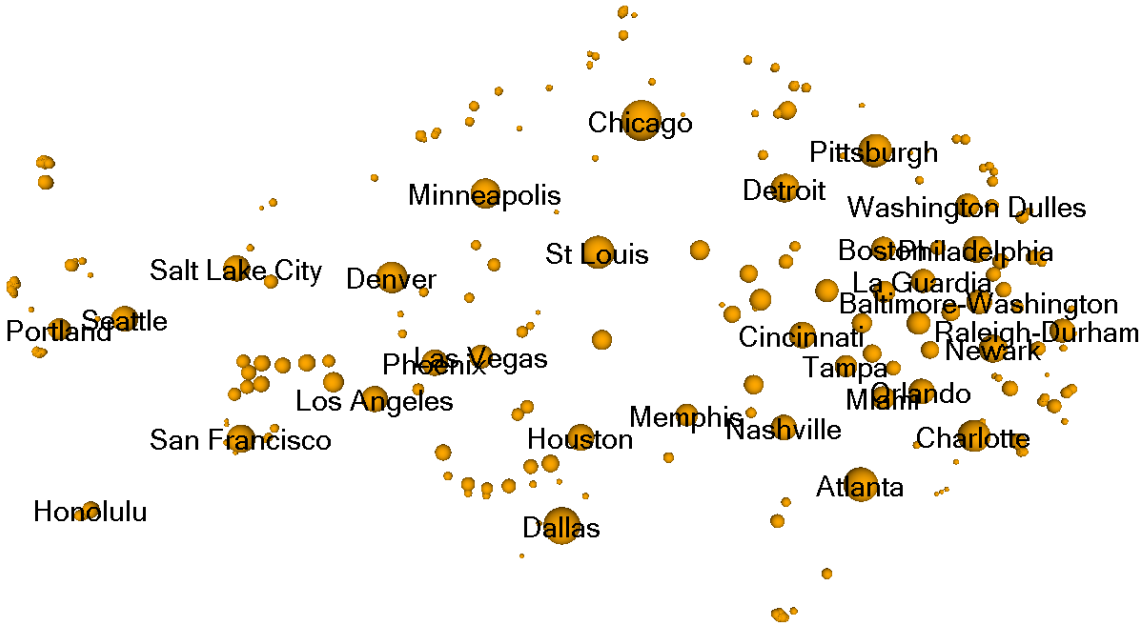
Figure 6.4: Layouts of the World Trade graph



(a) Layout with small Fruchterman-Reingold energy

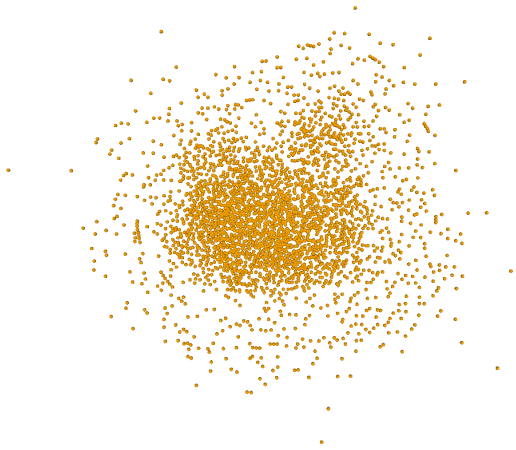


(b) Layout with small LinLog$^{\text{vert}}$ energy; normalized$^{\text{vert}}$ atedge length 0.085

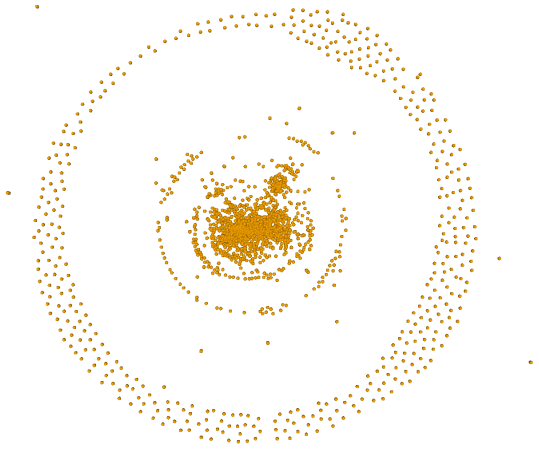


(c) Layout with small LinLog$^{\text{endv}}$ energy; normalized$^{\text{endv}}$ atedge length 0.419

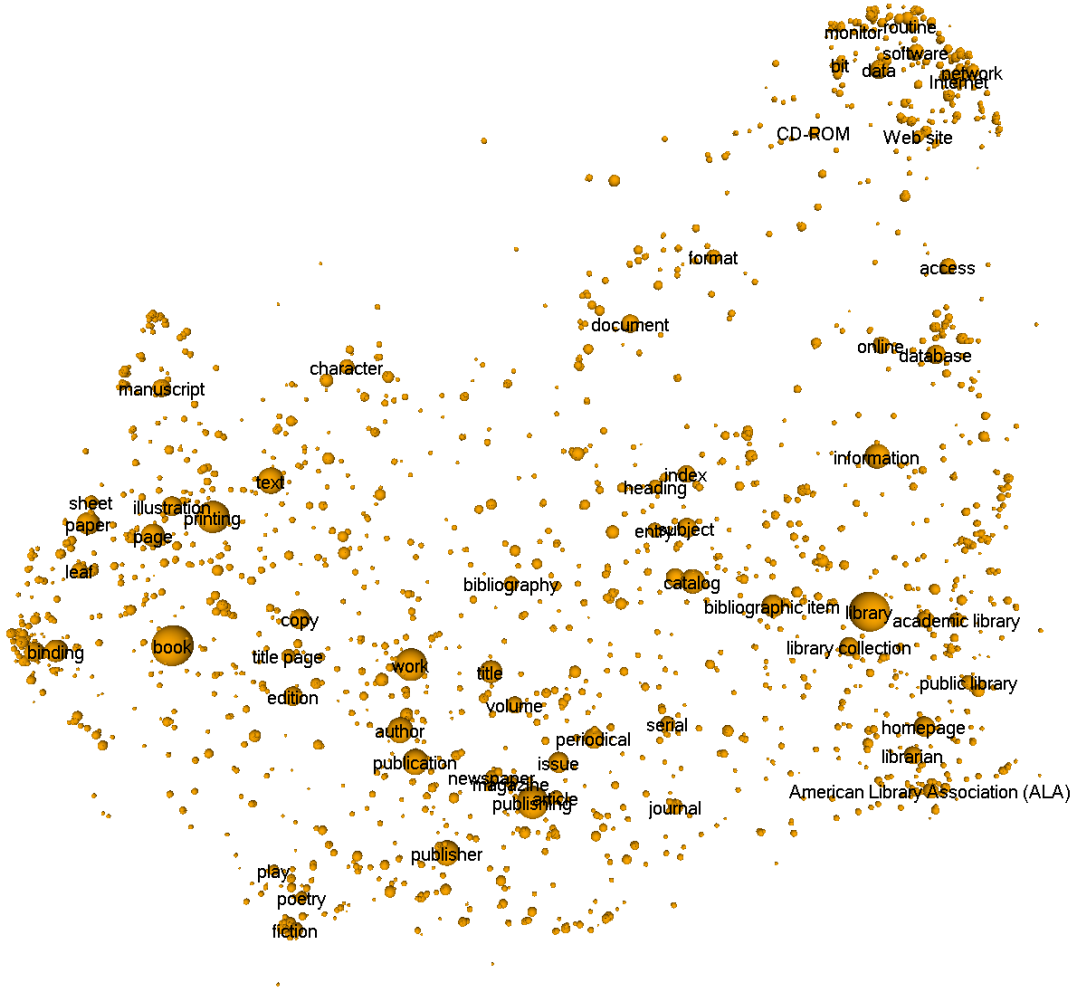
Figure 6.5: Layouts of the US Airlines graph. Some distant airports in Alaska and the South Sea (e.g. Guam) are omitted to improve readability.



(a) Layout with small Fruchterman-Reingold energy

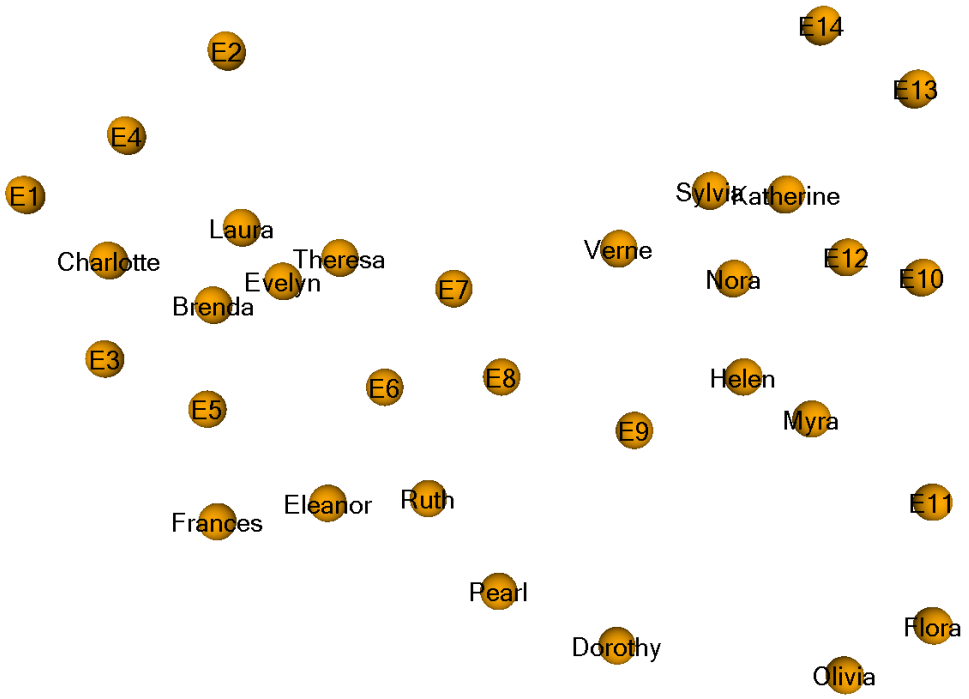


(b) Layout with small LinLog$^{\text{vert}}$ energy; normalized$^{\text{vert}}$ atedge length 0.179

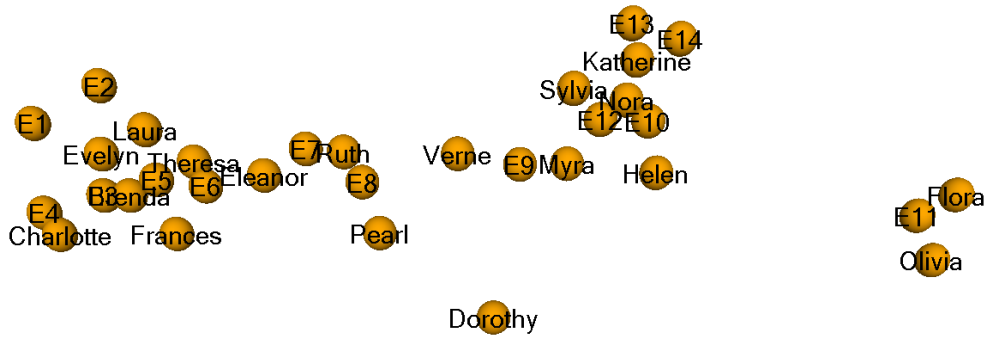


(c) Layout with small LinLog$^{\text{endv}}$ energy; normalized$^{\text{endv}}$ atedge length 0.341

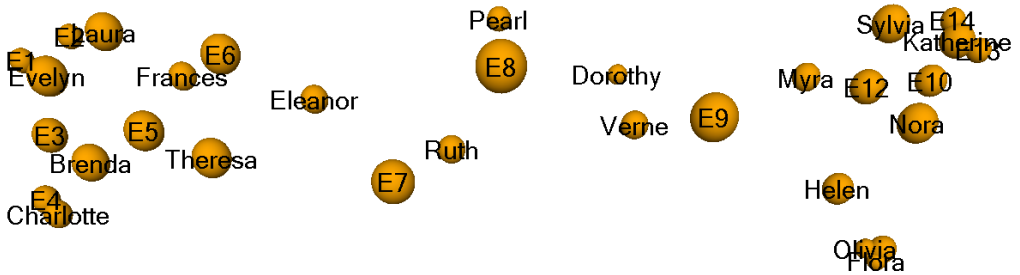
Figure 6.6: Layouts of the ODLIS graph



(a) Layout with small Fruchterman-Reingold energy

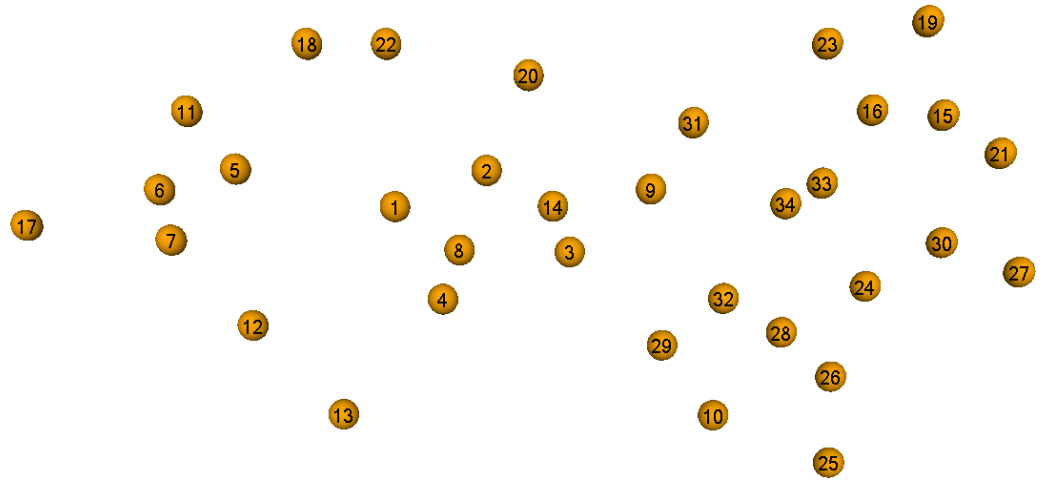


(b) Layout with small LinLog$^{\text{vert}}$ energy; normalized$^{\text{vert}}$ atedge length 0.399

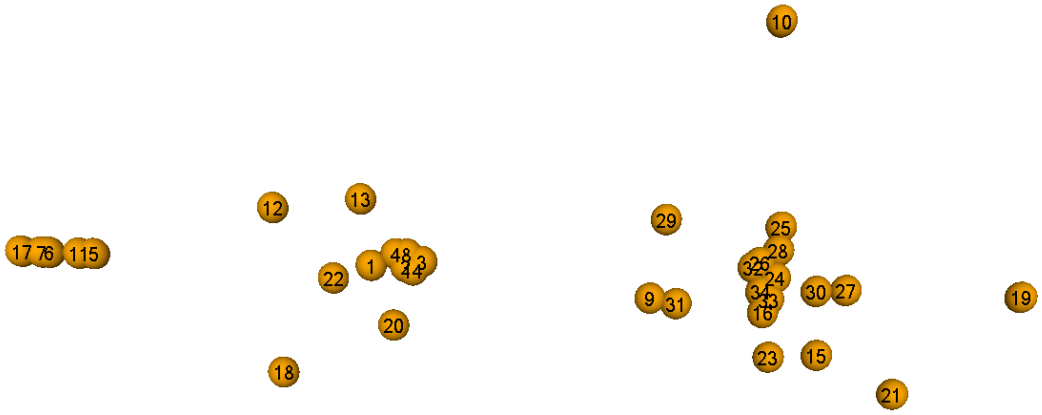


(c) Layout with small LinLog$^{\text{endv}}$ energy; normalized$^{\text{endv}}$ atedge length 0.430

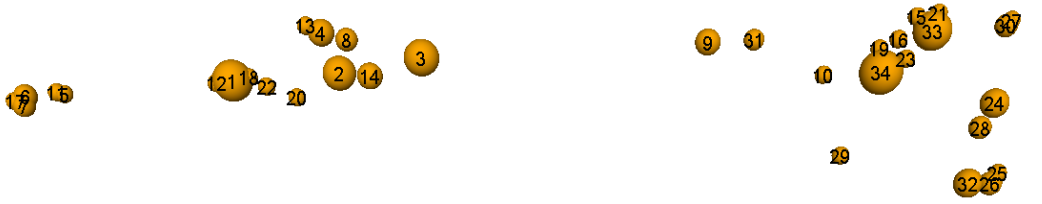
Figure 6.7: Layouts of the Southern Women graph



(a) Layout with small Fruchterman-Reingold energy

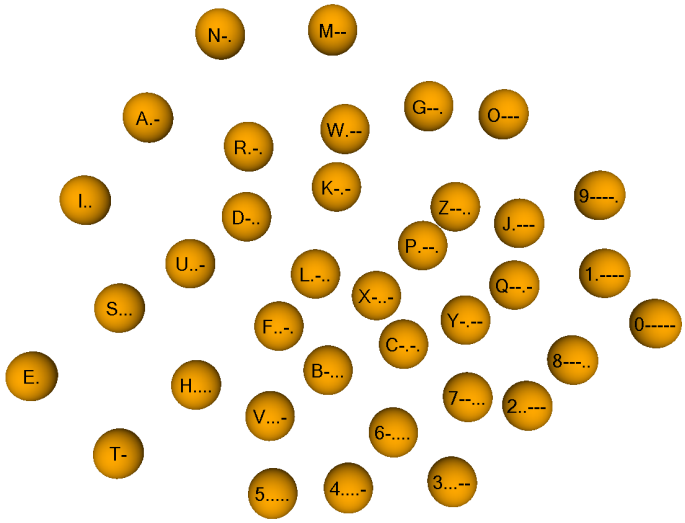


(b) Layout with small LinLog$^{\text{vert}}$ energy; normalized$^{\text{vert}}$ atedge length 0.251

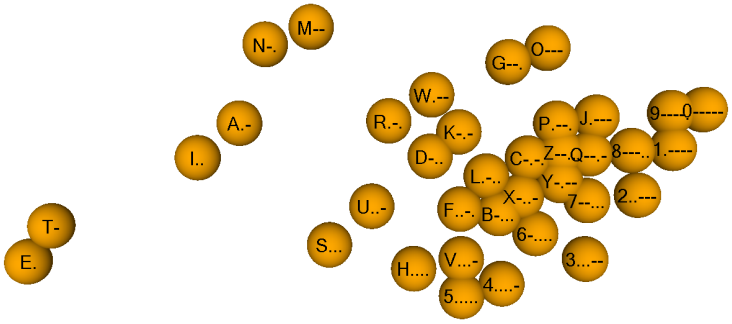


(c) Layout with small LinLog$^{\text{endv}}$ energy; normalized$^{\text{endv}}$ atedge length 0.276

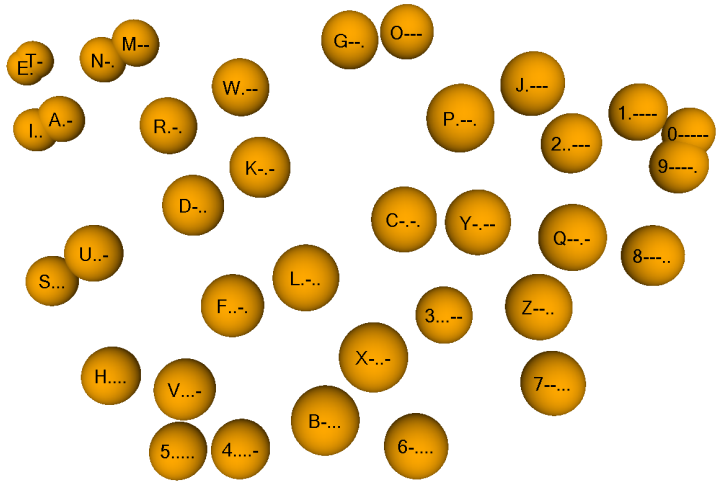
Figure 6.8: Layouts of the Karate Club graph



(a) Layout with small Fruchterman-Reingold energy

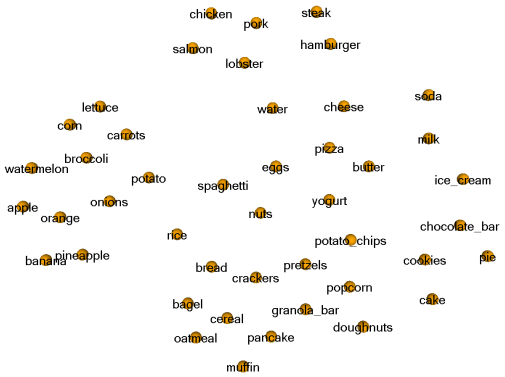


(b) Layout with small LinLog$^{\text{vert}}$ energy; normalized$^{\text{vert}}$ atedge length 0.620

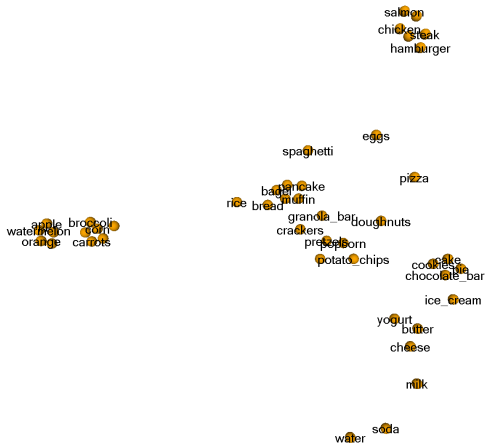


(c) Layout with small LinLog$^{\text{endv}}$ energy; normalized$^{\text{endv}}$ atedge length 0.765

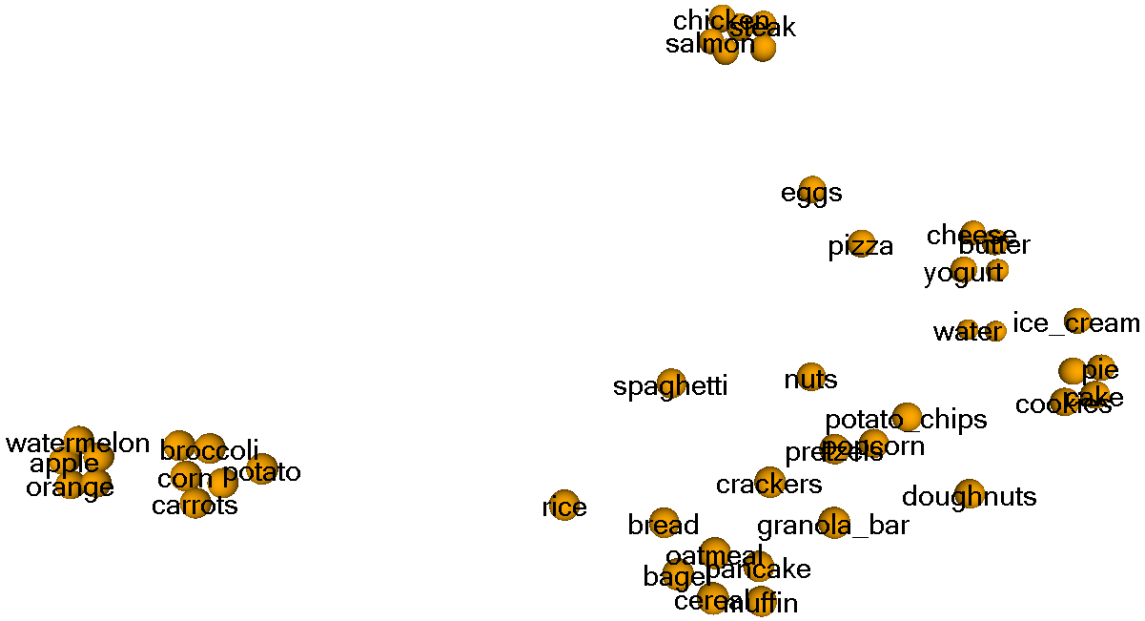
Figure 6.9: Layouts of the Morse Code Confusion graph



(a) Layout with small Fruchterman-Reingold energy

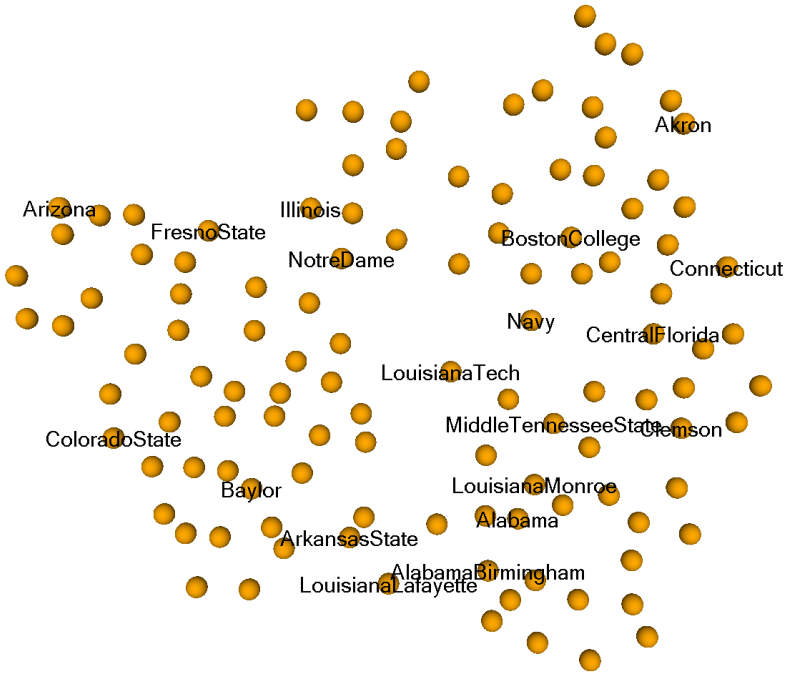


(b) Layout with small LinLog$^{\text{vert}}$ energy; normalized$^{\text{vert}}$ atedge length 0.431

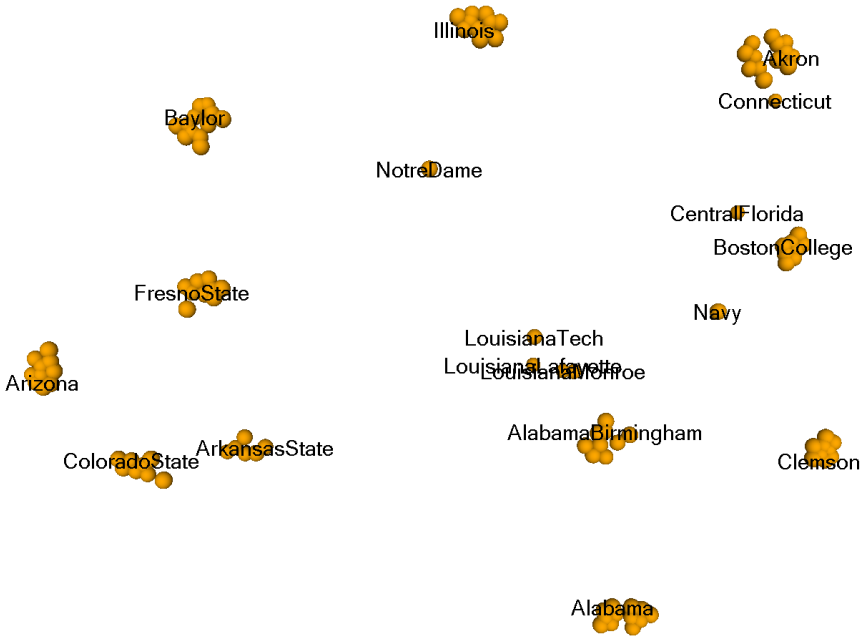


(c) Layout with small LinLog$^{\text{endv}}$ energy; normalized$^{\text{endv}}$ atedge length 0.444

Figure 6.10: Layouts of the Food Classification graph



(a) Layout with small Fruchterman-Reingold energy



(b) Layout with small LinLog$^{\text{endv}}$ energy; normalized$^{\text{endv}}$ atedge length 0.267. Because of the almost uniform vertex degrees, the LinLog$^{\text{vert}}$ layout is very similar.

Figure 6.11: Layouts of the College Football graph. Names are shown for one representative of each conference and all conference-independent teams. In Subfigure (b), vertex Middle Tennessee State is unnamed and very close to vertex Louisiana Monroe.

6.6 Summary

6.6.1 Main Results

- Layouts with optimal normalized atedge length tend to place densely connected vertices at the same position. The normalized atedge length can be parameterized to control this tendency. The resulting layout quality measure is called r-normalized atedge length, is equivalent to the normalized atedge length for $r=1$, and is biased against very small and very large atvertex distances for $r<1$ (Theorem 6.1).
- In the r-LinPoly energy model, adjacent vertices attract and all vertices repulse. Minima of this energy model are also minima of the r-normalized atedge length (Theorem 6.2). This enables the application of existing force calculation algorithms to compute layouts with small r-normalized atedge length.
- With the proof technique of Theorem 6.2, the optimal layouts of various force and energy models can be characterized as minimizing the ratio of the average atedge length to the average atvertex distance (for appropriate notions of "average"). This allows a uniform characterization of energy-based and spectral layouts, since spectral layouts are known to minimize similar ratios.
- Previous force and energy models (for graphs without vertex weights) are biased towards placing high-degree vertices on central positions. r-LinPoly$^{\text{endv}}$ is the first energy model without this bias.
- Previous force and energy models (like the Fruchterman-Reingold model) are strongly biased towards uniform edge lengths or uniform vertex distances. This may improve the readability of box-line visualizations, but impedes the separation of subgraphs with small inter-density. r-LinPoly is the first energy model that allows to control the bias towards uniform vertex distances (via the parameter r), and thus the separation of subgraphs. For example, the distance of subgraphs in 0-LinPoly (i.e. LinLog) layouts is roughly inversely proportional to their inter-density (Theorem 6.4 and its discussion).
- Empirical observations for real-world graphs confirm that the normalized$^{\text{endv}}$ atedge length can identify authoritative groupings better than the normalized$^{\text{vert}}$ atedge length for graphs with nonuniform vertex degrees, and that both measures identify authoritative groupings better than the Fruchterman-Reingold energy.

6.6.2 Measures

Layout quality measures for a graph $(\mathcal{V}, \mathcal{E})$ and a layout p.

Total atedge length:

$$\mathrm{Q}(\mathrm{p}) \;=\; \sum\nolimits_{\{v_1,v_2\}\in\mathcal{E}} \|\mathrm{p}(v_1)-\mathrm{p}(v_2)\|$$

- bias towards layouts with small atvertex distances (Theorem 3.1)
- bias towards graphs with small density (Theorem 3.1)

r-scaled atedge length (for $r \in \mathbb{R}$, $r \neq 0$):

$$\mathrm{Q}^{r\text{-scal}}(\mathrm{p}) \;=\; \Big(\sum\nolimits_{\{v_1,v_2\}\in\mathcal{E}} \|\mathrm{p}(v_1)-\mathrm{p}(v_2)\|\Big) \;/\; \Big(\sum\nolimits_{\{v_1,v_2\}\in\mathcal{V}^{\langle 2\rangle}} \|\mathrm{p}(v_1)-\mathrm{p}(v_2)\|^r\Big)^{1/r}$$

- if $r < 1$, bias towards layouts with uniform atvertex distances (Theorem 6.1);
  if $r = 1$, no bias towards any layout for graphs with uniform density (Theorem 6.1)
- bias towards graphs with small density (Theorem 3.1)

r-normalized atedge length (for $r \in \mathbb{R}$):

$$\mathrm{Q}^{r\text{-norm}}(\mathrm{p}) \;=\; \begin{cases} \dfrac{\sum_{\{v_1,v_2\}\in\mathcal{E}} \|\mathrm{p}(v_1)-\mathrm{p}(v_2)\|}{|\mathcal{E}|} \;/\; \left(\dfrac{\sum_{\{v_1,v_2\}\in\mathcal{V}^{\langle 2\rangle}} \|\mathrm{p}(v_1)-\mathrm{p}(v_2)\|^r}{|\mathcal{V}^{\langle 2\rangle}|}\right)^{1/r} & \text{if } r \neq 0 \\ \dfrac{\sum_{\{v_1,v_2\}\in\mathcal{E}} \|\mathrm{p}(v_1)-\mathrm{p}(v_2)\|}{|\mathcal{E}|} \;/\; \exp\!\left(\dfrac{\sum_{\{v_1,v_2\}\in\mathcal{V}^{\langle 2\rangle}} \ln\|\mathrm{p}(v_1)-\mathrm{p}(v_2)\|}{|\mathcal{V}^{\langle 2\rangle}|}\right) & \text{if } r = 0 \end{cases}$$

- if $r < 1$, bias towards layouts with uniform atvertex distances (Theorem 6.1);
  if $r = 1$, no bias towards any layout for graphs with uniform density (Theorem 6.1)
- no bias towards graphs with particular vertex weights or density (Theorem 3.1)

r-LinPoly energy (for $r \in \mathbb{R}$):

$$\mathrm{U}^r(\mathrm{p}) \;=\; \begin{cases} \sum_{\{v_1,v_2\}\in\mathcal{E}} \|\mathrm{p}(v_1)-\mathrm{p}(v_2)\| - \sum_{\{v_1,v_2\}\in\mathcal{V}^{\langle 2\rangle}} \frac{1}{r}\|\mathrm{p}(v_1)-\mathrm{p}(v_2)\|^r & \text{if } r \neq 0 \\ \sum_{\{v_1,v_2\}\in\mathcal{E}} \|\mathrm{p}(v_1)-\mathrm{p}(v_2)\| - \sum_{\{v_1,v_2\}\in\mathcal{V}^{\langle 2\rangle}} \ln\|\mathrm{p}(v_1)-\mathrm{p}(v_2)\| & \text{if } r = 0 \end{cases}$$

- for a fixed graph and $r < 1$, minima of the r-LinPoly energy are also minima of the r-normalized atedge length (Theorem 6.2)

Chapter 7

Quality Measures for Graph Clusterings II

Chapter 3 introduced and normalized the total atedge length as quality measure for graph assignments. This chapter introduces and partly normalizes an alternative quality measure, which will not be applied to graph orderings and graph layouts, and is therefore only defined for graph clusterings. While the total atedge length and its normalized variants formalize that different clusters should be sparsely connected, the measures of this chapter formalize that different clusters should not be connected at all. The two sections of this chapter define the measures and discuss related work; applications are treated extensively in Chapters 11 and 12.

7.1 Definitions and Visualization

This section introduces and analyzes an absolute quality measure for graph clusterings, and a normalization of this measure. It shows how the values of the two measures are reflected in graph visualizations, and gives examples. As before, smaller values of the quality measures indicate better clusterings.

7.1.1 The Total Atpair Cut

Definition The *total atpair cut* of a graph $(\mathcal{V}, \mathcal{E})$ in a clustering p is

$$\mathrm{P}(\mathrm{p}) \;:=\; \sum\nolimits_{\{v_1,v_2\}\in\mathcal{V}^{\langle 2\rangle}:\ \mathrm{p}(v_1)\neq\mathrm{p}(v_2),\ \{\mathrm{p}(v_1),\mathrm{p}(v_2)\}\in\mathrm{p}(\mathcal{E})} 1 \ .$$

While the total atedge cut was defined in Section 4.1 as the number of atedges between different connected clusters, the total atpair cut is the number of atvertex pairs (or shortly atpairs) between different connected clusters.

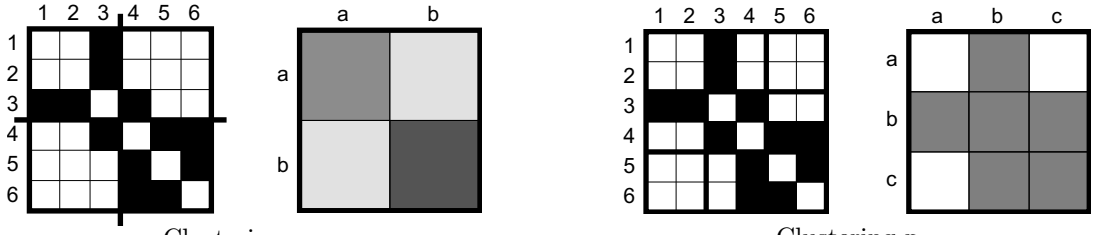
In contrast to the total atedge cut, the total atpair cut does not depend on the number of atedges, but only on the existence of (at)edges between different clusters. If each cluster has unit weight, the total atpair cut is simply the number of non-loop edges of the cluster graph. In general, each non-loop cluster edge contributes the

weight product of its two cluster vertices to the total atpair cut. Thus the total atpair cut equals the total atedge cut if every pair of atvertices in different clusters is connected by exactly one atedge.

Visualization In matrix visualizations, the atvertex pairs between two different clusters are represented by two so-called *cluster-pair rectangles*, which are fatly bounded rectangles in visualizations of the base graph, and single matrix elements in visualizations of the cluster graph. The number of atvertex pairs between the clusters equals the area of each of the corresponding cluster-pair rectangles. If the two clusters are connected by at least one edge, then the corresponding cluster-pair rectangles are not white. Thus the total atpair cut equals the total area of the non-white cluster-pair rectangles above the diagonal (or equivalently below the diagonal), in matrix visualizations of both the base graph and the cluster graph. It is more explicit in visualizations of the cluster graph, where non-white cluster-pair rectangles are *entirely* non-white, and thus the total atpair cut equals the total non-white area above the diagonal.

The representation of the total atpair cut is similar to the representation of the total atedge cut, which equals the total color weight in the cluster-pair rectangles above the diagonal.

Examples The left clustering in Figure 7.1 has a total atpair cut of 9, and the right clustering has a total atpair cut of 8. Accordingly, the non-white area above the diagonal is slightly smaller in the visualizations of the right cluster graph. In contrast, the color weight above the diagonal is much greater in the visualizations of the right cluster graph, because the right clustering has a greater total atedge cut.



Clustering p_1,
with $P(p_1) = 9$ and $P^{scal}(p_1) = \frac{9}{9} = 1$

Clustering p_2,
with $P(p_2) = 8$ and $P^{scal}(p_2) = \frac{8}{12} \approx 0.67$

Figure 7.1: Clusterings of a graph with unit vertex weights and unit edge weights

7.1.2 Bias of the Total Atpair Cut

The clustering quality measures of Chapter 3 formalize that different clusters should have a small inter-density. Accordingly, it was verified that they take the same value for all clusterings of any graph with uniform density. The clustering quality measures in this chapter formalize that different clusters should not be connected at all. Unfortunately, there are no non-trivial graphs where all clusterings satisfy this criterion equally well, and thus a similar validation as in Chapter 3 will not be performed in this chapter.

Informally, the total atpair cut is affected by the following three properties of graphs and clusterings. First, it prefers graphs with few atvertices: Doubling all vertex weights increases the total atpair cut by a factor of 4. This bias will be removed by the normalization in the next subsection. Second, the total atpair cut prefers graphs with few edges: Removing an edge never increases and sometimes decreases the total atpair cut. Finally, the total atpair cut prefers clusterings with certain granularities: The trivial clustering with one cluster has the minimum total atpair cut of 0, and fine-grained clusterings tend to have a small total atpair cut because the probability that clusters are connected generally grows with their size.

7.1.3 The Scaled Atpair Cut

Definition The scaled atpair cut is obtained by normalizing the total atpair cut, i.e. the number of atvertex pairs between *connected* different clusters, with the number of atvertex pairs between *all* different clusters. Formally, the *scaled atpair cut* of a graph $(\mathcal{V}, \mathcal{E})$ in a clustering p is

$$\mathrm{P}^{\mathrm{scal}}(\mathrm{p}) := \frac{\mathrm{P}(\mathrm{p})}{\sum_{\{v_1,v_2\}\in\mathcal{V}^{\langle 2\rangle}\colon\, \mathrm{p}(v_1)\neq\mathrm{p}(v_2)} 1} = \frac{\sum_{\{v_1,v_2\}\in\mathcal{V}^{\langle 2\rangle}\colon\, \mathrm{p}(v_1)\neq\mathrm{p}(v_2),\ \{\mathrm{p}(v_1),\mathrm{p}(v_2)\}\in\mathrm{p}(\mathcal{E})} 1}{\sum_{\{v_1,v_2\}\in\mathcal{V}^{\langle 2\rangle}\colon\, \mathrm{p}(v_1)\neq\mathrm{p}(v_2)} 1}\,.$$

The same normalization was applied to the total atedge cut to derive the scaled atedge cut in Section 3.1.3.

In contrast to the total atpair cut, the scaled atpair cut is not biased towards graphs with few atvertices. The values of the scaled atpair cut are clearly interpretable as the fraction of all inter-cluster atvertex pairs that belong to connected clusters. The minimum possible value of 0 means that there are no inter-cluster edges and the maximum possible value of 1 means that all pairs of different clusters are connected. Additional benefits of the scaled atpair cut are its analogy to the scaled atedge cut, and its clear representation in matrix visualizations particularly of cluster graphs.

A normalized atpair cut (in analogy to the normalized atedge cut) will not be defined because a further normalization with $\frac{|\mathcal{E}|}{|\mathcal{V}^{\langle 2\rangle}|}$ destroys the mentioned benefits.

Visualization In matrix visualizations of both the base graph and the cluster graph, the total atpair cut equals the total area of the non-white cluster-pair rectangles above the diagonal. The number of inter-cluster atvertex pairs equals the total area of the cluster-pair rectangles above the diagonal. The scaled atpair cut is the quotient of these two terms. It is more explicit in visualizations of the cluster graph, where it equals the fraction of non-white area in the off-diagonal area.

Examples The left clustering in Figure 7.1 has a scaled atpair cut of 1, and accordingly the entire off-diagonal area is non-white in the matrix visualization of the cluster graph. The right clustering has a scaled atpair cut of $\frac{8}{12} = \frac{2}{3}$, and thus one third of the off-diagonal area is white. For the clustering where each cluster contains exactly one vertex, the scaled atpair cut is $\frac{6}{15} = \frac{2}{5}$.

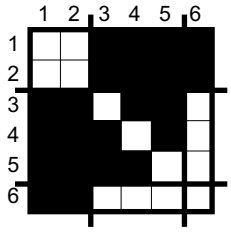
7.2 Related Work

The total atpair cut and the scaled atpair cut formalize that different clusters should not be connected by edges. Similar notions of a cluster have been proposed for social networks in positional analysis and blockmodeling. In this context, clusters are often denoted as blocks, positions, or roles, and clusterings as role assignments. This section is based on surveys of positional analysis by Lerner [Ler05] and Wasserman and Faust [WF94, Chapters 9, 12].

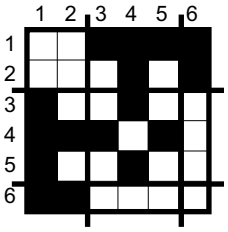
Definition Let (V, E) be a graph without vertex and edge weights, and let p be a clustering of this graph.

Then the clustering p is *structural* if each two vertices in the same cluster are connected to the same other vertices, i.e. if for all vertices $v_1, v_2 \in V$ with $\mathrm{p}(v_1) = \mathrm{p}(v_2)$ holds $\mathrm{N}(v_1) \setminus \{v_2\} = \mathrm{N}(v_2) \setminus \{v_1\}$ (where $\mathrm{N}(v)$ is the set of neighbors of v).

The clustering p is *regular* if vertices in the same cluster are connected to the same clusters, i.e. if for all vertices $v_1, v_2 \in V$ with $\mathrm{p}(v_1) = \mathrm{p}(v_2)$ holds $\mathrm{p}(\mathrm{N}(v_1)) = \mathrm{p}(\mathrm{N}(v_2))$.



Structural clustering



Regular clustering

Figure 7.2: Clusterings of two graphs without vertex weights and edge weights

Examples The left clustering in Figure 7.2 is both structural and regular. The right clustering in Figure 7.2 is regular but not structural. Trivial clusterings where each cluster contains exactly one vertex are both structural and regular.

Discussion In a regular or structural clustering, each pair of clusters is either not connected or fairly densely connected. The scaled atpair cut is similar in that it rewards pairs of clusters that are not connected (but only pairs of different clusters); it is different in that it ignores how clusters are connected if they are connected.

If a clustering is not regular, then at least one of the clusters contains two vertices that are not connected to the same clusters. Repeated splitting of such clusters finally produces a regular clustering. The splits usually reduce the scaled atpair cut, and thus the resulting regular clustering may have a relatively small scaled atpair cut, compared to other clusterings of the same graph with similar granularity.

Most real-world graphs have many regular clusterings, but hardly any nontrivial structural clusterings, so structural clusterings are practically relevant mainly as ideals that can be approximated to a certain degree.

7.3 Summary

7.3.1 Main Results

- The total atpair cut is the number of atvertex pairs between different connected clusters, in analogy to the total atedge cut, which is the number of atedges between different (connected) clusters.
- The scaled atpair cut equals the ratio of atvertex pairs between different connected clusters to all inter-cluster atvertex pairs. The minimum possible value of 0 means that there are no inter-cluster edges, and the maximum possible value of 1 means that all pairs of different clusters are connected by at least one edge.
- The total atpair cut is biased towards graphs with few atvertices and towards clusterings with certain granularities. Both the total atpair cut and the scaled atpair cut tend to be smaller for graphs with few edges.
- In matrix visualizations of the cluster graph, the total atpair cut equals the non-white area above the diagonal, and the scaled atpair cut equals the fraction of non-white area above (or outside) the diagonal. That is, the total atpair cut corresponds to the non-white area in the same way as the total atedge cut corresponds to the color weight.

7.3.2 Measures for Graphs

Clustering quality measures for a base graph $G = (\mathcal{V}, \mathcal{E})$, a clustering p, and the resulting cluster graph $\mathrm{p}(G) = (\mathcal{V}', \mathcal{E}')$.

Total atpair cut:

$$\begin{aligned} \mathrm{P}(\mathrm{p}) &= \sum\nolimits_{\{v_1,v_2\}\in\mathcal{V}^{\langle 2\rangle}\colon\ \mathrm{p}(v_1)\neq\mathrm{p}(v_2),\ \{\mathrm{p}(v_1),\mathrm{p}(v_2)\}\in\mathrm{p}(\mathcal{E})} 1 \\ &= \sum\nolimits_{\{v'_1,v'_2\}\in\mathcal{V}'^{\langle 2\rangle}\colon\ \{v'_1,v'_2\}\in\mathcal{E}'} 1 \end{aligned}$$

- bias towards graphs with few atvertices

Scaled atpair cut:

$$\begin{aligned} \mathrm{P}^{\mathrm{scal}}(\mathrm{p}) &= \frac{\mathrm{P}(\mathrm{p})}{\sum_{\{v_1,v_2\}\in\mathcal{V}^{\langle 2\rangle}\colon\ \mathrm{p}(v_1)\neq\mathrm{p}(v_2)} 1} \\ &= \frac{\mathrm{P}(\mathrm{p})}{\sum_{\{v'_1,v'_2\}\in\mathcal{V}'^{\langle 2\rangle}} 1} \end{aligned}$$

- no bias towards graphs with few atvertices

7.3.3 Measures for Graphs without Vertex Weights

Clustering quality measures for a base graph without vertex weights $G = (V, \mathcal{E})$ with vertex degrees deg, a clustering p, and the resulting cluster graph $\mathrm{p}(G) = ((V', \mathrm{w}'), \mathcal{E}')$ with vertex degrees deg′. The quality measures are derived by transforming G into a graph with vertex weights using either unit vertex weights or unit endvertex weights, and applying the quality measures for graphs with vertex weights to the transformed graph.

Total atpair$^{\mathrm{vert}}$ cut:

$$\begin{aligned} \mathrm{P}^{\mathrm{vert}}(\mathrm{p}) &= \sum\nolimits_{\{v_1,v_2\}\in V^{(2)}:\ \mathrm{p}(v_1)\neq\mathrm{p}(v_2),\ \{\mathrm{p}(v_1),\mathrm{p}(v_2)\}\in\mathrm{p}(\mathcal{E})} 1 \\ &= \sum\nolimits_{\{v'_1,v'_2\}\in(V',\mathrm{w}')^{\langle 2\rangle}:\ \{v'_1,v'_2\}\in\mathcal{E}'} 1 \end{aligned}$$

- bias towards graphs with few vertices

Scaled atpair$^{\mathrm{vert}}$ cut:

$$\begin{aligned} \mathrm{P}^{\mathrm{scalvert}}(\mathrm{p}) &= \frac{\mathrm{P}(\mathrm{p})}{\sum_{\{v_1,v_2\}\in V^{(2)}:\ \mathrm{p}(v_1)\neq\mathrm{p}(v_2)} 1} \\ &= \frac{\mathrm{P}(\mathrm{p})}{\sum_{\{v'_1,v'_2\}\in(V',\mathrm{w}')^{\langle 2\rangle}} 1} \end{aligned}$$

- no bias towards graphs with few vertices

Total atpair$^{\mathrm{endv}}$ cut:

$$\begin{aligned} \mathrm{P}^{\mathrm{endv}}(\mathrm{p}) &= \sum\nolimits_{\{v_1,v_2\}\in(V,\deg)^{\langle 2\rangle}:\ \mathrm{p}(v_1)\neq\mathrm{p}(v_2),\ \{\mathrm{p}(v_1),\mathrm{p}(v_2)\}\in\mathrm{p}(\mathcal{E})} 1 \\ &= \sum\nolimits_{\{v'_1,v'_2\}\in(V',\deg')^{\langle 2\rangle}:\ \{v'_1,v'_2\}\in\mathcal{E}'} 1 \end{aligned}$$

- bias towards graphs with few endvertices

Scaled atpair$^{\mathrm{endv}}$ cut:

$$\begin{aligned} \mathrm{P}^{\mathrm{scalendv}}(\mathrm{p}) &= \frac{\mathrm{P}(\mathrm{p})}{\sum_{\{v_1,v_2\}\in(V,\deg)^{\langle 2\rangle}:\ \mathrm{p}(v_1)\neq\mathrm{p}(v_2)} 1} \\ &= \frac{\mathrm{P}(\mathrm{p})}{\sum_{\{v'_1,v'_2\}\in(V',\deg')^{\langle 2\rangle}} 1} \end{aligned}$$

- no bias towards graphs with few endvertices

Chapter 8

Conclusion and Discussion

8.1 Unification

The first goal of Part I was the unification of quality measures for graph clusterings, graph orderings, and graph layouts. Transformations and generalizations were introduced that allow to join many new and existing quality measures into a single class of measures, called r-normalized s-atedge length (see Table 8.1). The specific transformations will be discussed below; the general benefits are as follows:

- To select an appropriate quality measure for a specific application, users are not required to compare a large number of seemingly unrelated measures, but only to provide values for few parameters with well-defined meanings: The parameters r and s of the r-normalized s-atedge length control the bias towards uniform atvertex distances and uniform atedge lengths, respectively (see Section 6.1); the choice between unit vertex weights and unit endvertex weights, which is only required for graphs without vertex weights, determines whether there is a bias towards placing high-degree vertices on central positions (see Subsection 3.1.5).
- The evaluation of many specific quality measures can be partly replaced with an evaluation of a single class of quality measures, and an evaluation of the impact of its parameters. For example, Section 3.1 analyzed quality measures for graph assignments without distinguishing between clusterings, orderings, and layouts.
- The r-normalized s-atedge length includes many measures that have not yet been proposed in the literature, but may be useful. For example, some existing layout quality measures for graphs without vertex weights have only been proposed in a variant that assumes unit vertex weight; a variant that assumes unit endvertex weights may also be valuable, as it is for many other measures.

Among the assignment quality measures for graphs without vertex weights, there are many pairs of two similar measures, of which one implicitly assumes unit vertex weights and the other implicitly assumes unit endvertex weights. Each of these pairs can be unified into a single measure for graphs with vertex weights.

scaled$^{\text{vert}}$ atedge cut f. clusterings	scaled$^{\text{endv}}$ atedge cut f. clusterings	total atedge length f. orderings$^{\text{vert}}$	total atedge length f. orderings$^{\text{endv}}$	LinLog$^{\text{vert}}$ energy model for layouts	LinLog$^{\text{endv}}$ energy model for layouts	Several other measures
scaled atedge cut for graph clusterings $\frac{\sum_{\mathcal{E}} \overline{\delta}(\mathrm{p}(v_1),\mathrm{p}(v_2))}{\sum_{\mathcal{V}^{\langle 2\rangle}} \overline{\delta}(\mathrm{p}(v_1),\mathrm{p}(v_2))}$		total atedge length for graph orderings $\sum_{\mathcal{E}} \lvert\mathrm{p}(v_1) - \mathrm{p}(v_2)\rvert$		LinLog energy model for graph layouts $\sum_{\mathcal{E}} \lVert\mathrm{p}(v_1)-\mathrm{p}(v_2)\rVert - \sum_{\mathcal{V}^{\langle 2\rangle}} \ln \lVert\mathrm{p}(v_1)-\mathrm{p}(v_2)\rVert$		$\vdots$
normalized atedge cut for graph clusterings $\frac{\sum_{\mathcal{E}} \overline{\delta}(\mathrm{p}(v_1),\mathrm{p}(v_2))/\lvert\mathcal{E}\rvert}{\sum_{\mathcal{V}^{\langle 2\rangle}} \overline{\delta}(\mathrm{p}(v_1),\mathrm{p}(v_2))/\lvert\mathcal{V}^{\langle 2\rangle}\rvert}$		normalized atedge length for graph orderings $\frac{\sum_{\mathcal{E}} \lvert\mathrm{p}(v_1)-\mathrm{p}(v_2)\rvert/\lvert\mathcal{E}\rvert}{\sum_{\mathcal{V}^{\langle 2\rangle}} \lvert\mathrm{p}(v_1)-\mathrm{p}(v_2)\rvert/\lvert\mathcal{V}^{\langle 2\rangle}\rvert}$		0-normalized atedge length for graph layouts $\frac{\sum_{\mathcal{E}} \lVert\mathrm{p}(v_1)-\mathrm{p}(v_2)\rVert/\lvert\mathcal{E}\rvert}{\left(\prod_{\mathcal{V}^{\langle 2\rangle}} \lVert\mathrm{p}(v_1)-\mathrm{p}(v_2)\rVert\right)^{1/\lvert\mathcal{V}^{\langle 2\rangle}\rvert}}$		$\vdots$
normalized atedge length for graph assignments $\frac{\sum_{\mathcal{E}} \mathrm{d}(\mathrm{p}(v_1),\mathrm{p}(v_2))/\lvert\mathcal{E}\rvert}{\sum_{\mathcal{V}^{\langle 2\rangle}} \mathrm{d}(\mathrm{p}(v_1),\mathrm{p}(v_2))/\lvert\mathcal{V}^{\langle 2\rangle}\rvert}$		normalized atedge length for graph assignments $\frac{\sum_{\mathcal{E}} \mathrm{d}(\mathrm{p}(v_1),\mathrm{p}(v_2))/\lvert\mathcal{E}\rvert}{\sum_{\mathcal{V}^{\langle 2\rangle}} \mathrm{d}(\mathrm{p}(v_1),\mathrm{p}(v_2))/\lvert\mathcal{V}^{\langle 2\rangle}\rvert}$		0-normalized atedge length for graph assignments $\frac{\sum_{\mathcal{E}} \mathrm{d}(\mathrm{p}(v_1),\mathrm{p}(v_2))/\lvert\mathcal{E}\rvert}{\left(\prod_{\mathcal{V}^{\langle 2\rangle}} \mathrm{d}(\mathrm{p}(v_1),\mathrm{p}(v_2))\right)^{1/\lvert\mathcal{V}^{\langle 2\rangle}\rvert}}$		$\vdots$
r-normalized s-atedge length for graph assignments $\frac{s\text{-th power mean of the atedge lengths}}{r\text{-th power mean of the atvertex distances}} = \frac{\left(\sum_{\{v_1,v_2\}\in\mathcal{E}} \mathrm{d}(\mathrm{p}(v_1),\mathrm{p}(v_2))^s/\lvert\mathcal{E}\rvert\right)^{1/s}}{\left(\sum_{\{v_1,v_2\}\in\mathcal{V}^{\langle 2\rangle}} \mathrm{d}(\mathrm{p}(v_1),\mathrm{p}(v_2))^r/\lvert\mathcal{V}^{\langle 2\rangle}\rvert\right)^{1/r}}$						

Table 8.1: Unification of quality measures for graph clusterings, graph orderings, and graph layouts, illustrated for six example measures. The first three transformations (from top to bottom) correspond to the three conclusions in the text.

Several well-known and new quality measures can be represented as ratio of the mean atedge length to the mean atvertex distance (for appropriate power means). These measures include the normalized atedge cut for clusterings (by definition), the total atedge length for orderings (Lemma 5.2)[1], the LinLog energy model for layouts (Theorem 6.2), the Fruchterman-Reingold force model for layouts (Section 6.4), the eigenvector-motivated Hall-Fiedler measure for layouts (Section 6.4)[2], and others (Sections 5.2 and 6.4) – measures whose original formulations are vastly different (see Figure 8.2).

This result not only relates several measures, but also shows that their optimization indeed groups densely connected vertices (by minimizing the mean atedge length)

[1]For orderings of a fixed graph, the total atedge length and the normalized atedge length (i.e. its representation as distance ratio) differ only by constant factor.

[2]For a fixed graph, each globally optimal layout of the three mentioned layout quality measures is also a global optimum of the corresponding distance ratio. Note that the transformation of the Hall-Fielder measure is not a new result of this work.

Normalized atedge cut for clusterings (Subsections 3.1.4, 4.5.2):

$$\frac{\sum_{\{v_1,v_2\}\in\mathcal{E}:\,\mathrm{p}(v_1)\neq\mathrm{p}(v_2)} 1}{\sum_{\{v_1,v_2\}\in\mathcal{V}^{\langle 2\rangle}:\,\mathrm{p}(v_1)\neq\mathrm{p}(v_2)} 1} \Big/ \frac{|\mathcal{E}|}{|\mathcal{V}^{\langle 2\rangle}|}$$

Total atedge length for orderings (Subsections 3.1.1, 5.4.2):

$$\sum_{\{v_1,v_2\}\in\mathcal{E}} |\mathrm{p}(v_1) - \mathrm{p}(v_2)|$$

LinLog energy model for layouts (Sections 6.2):

$$\sum_{\{v_1,v_2\}\in\mathcal{E}} \|\mathrm{p}(v_1)-\mathrm{p}(v_2)\| \;-\; \sum_{\{v_1,v_2\}\in\mathcal{V}^{\langle 2\rangle}} \ln \|\mathrm{p}(v_1)-\mathrm{p}(v_2)\|$$

Fruchterman-Reingold forces for layouts of unweighted graphs [FR91]:

$$\text{Between adjacent vertices: } \|\mathrm{p}(v_1)-\mathrm{p}(v_2)\|^2 \cdot \overrightarrow{\mathrm{p}(v_1)\mathrm{p}(v_2)}$$
$$\text{Between any two vertices: } \|\mathrm{p}(v_1)-\mathrm{p}(v_2)\|^{-1} \cdot \overrightarrow{\mathrm{p}(v_2)\mathrm{p}(v_1)}$$

Eigenvector-based layouts of graphs without vertex weights [Hal70]:

$$\text{Minimize } \sum_{\{v_1,v_2\}\in\mathcal{E}} \|\mathrm{p}(v_1)-\mathrm{p}(v_2)\|^2 \text{ subject to } \sum_{v\in V} \mathrm{p}(v)^2 = 1.$$

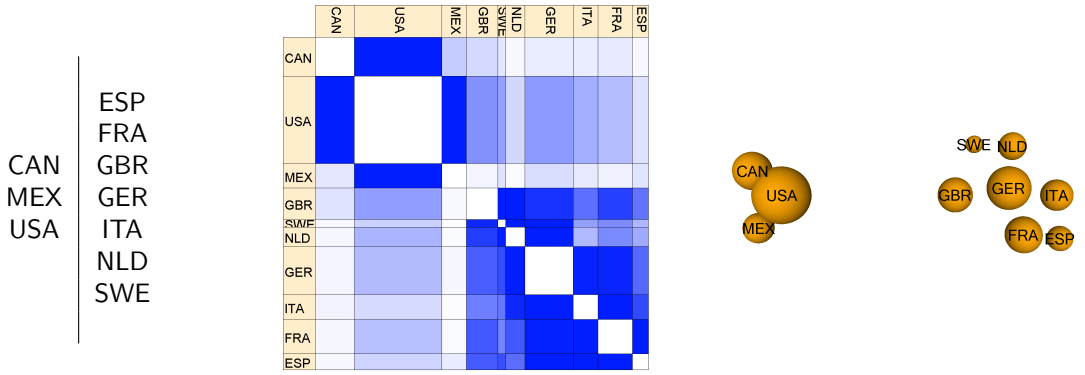
Table 8.2: Original formulations of five assignment quality measures that can be transformed into ratios of the mean atedge length to the mean atvertex distance

and separates sparsely connected vertices (by maximizing the mean atvertex distance), with varying biases towards uniform atedge lengths or atvertex distances. And it enables the application of existing algorithms to minimize distance ratios (e.g. algorithms for the Minimum Linear Arrangement problem, force calculation algorithms, and eigenvector-based algorithms).

Of course, there are many measures which have not been transformed into a ratio of the mean atedge length to the mean atvertex distance. A prominent example is Newman's modularity, which interestingly is equivalent to a *difference* of the mean atedge length and the mean atvertex distance (see the discussion of the shifted$^{\mathrm{endv}}$ cut in Subsection 4.3.4).

Graph clusterings, graph orderings, and graph layouts can be unified into graph assignments (mappings of vertices to a metric space), and thus can be evaluated with the same quality measures.

If identical or very similar quality measures are optimized to compute a clustering, an ordering, and a layout of the same graph, then the three resulting representations can be consistent (see Figure 8.1 and Section 3.3). However, this consistency is not guaranteed for all quality measures. For example, the optimization of the shifted atedge length generally produces nontrivial clusterings, but layouts with infinite distances (see Subsection 4.2.2). Even if the optimal clusterings, orderings, and layouts of a quality measure are consistent, as for the normalized atedge length, it



(a) Clustering (norm. atedge length)

(b) Ordering (normalized atedge length)

(c) Layout (0-normalized atedge length)

Figure 8.1: Consistent representations of the American-European Trade graph

may still be preferable to use variations of this measure; for example, the optimal layouts of the normalized atedge length generally place many vertices at the same position, and therefore most layouts in this work have been computed by optimizing the 0-normalized atedge length (see Subsection 6.1.2).

8.2 Validation

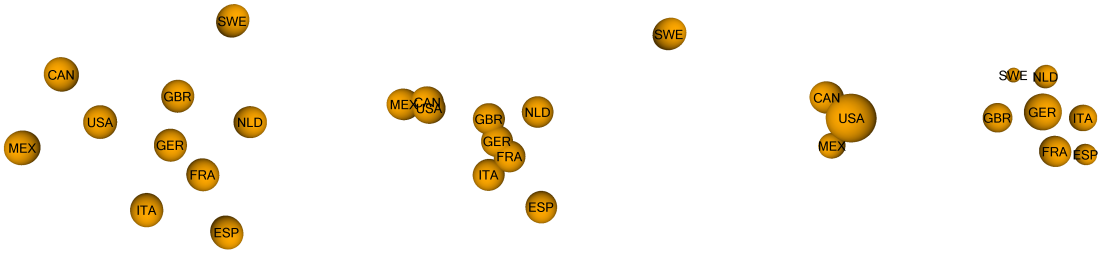
The second goal of Part I was to evaluate theoretically whether existing quality measures for graph assignments are internally valid – i.e. quantify how well densely connected vertices are grouped and sparsely connected vertices are separated[3] – and to derive new quality measures that are internally valid. Measures were evaluated and derived primarily through the detection and removal of biases, i.e. of preferences towards certain assignments even for graphs with uniform density. An extensive empirical evaluation of internal or external validity was not performed; however, the unification and theoretical evaluation is expected to facilitate future empirical evaluations, because fewer measures und fewer properties need to be examined.

Many existing quality measures for graph clusterings, graph orderings, and graph layouts are biased, e.g. towards grouping high-degree vertices, towards coarse-grained clusterings, or towards layouts with uniform edge lengths (see Sections 4.3, 5.2, 6.4).

Biased quality measures are not necessarily useless, because certain biases may be desirable for specific applications. However, knowing their biases is essential for interpreting their values and their optimal assignments, which generally reflect other graph properties besides density.

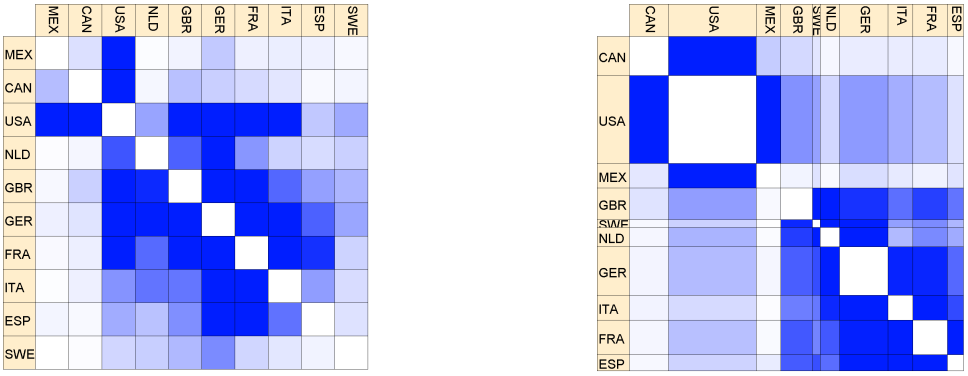
The normalized atedge length is an unbiased quality measure for graph assignments. It has been derived as quotient of the actual total atedge length and the expected total atedge length in the case of uniform density.

[3] As noted before, this is only one specific notion of assignment quality, other notions are possible but outside the scope of this work.



(a) Fruchterman-Reingold (b) LinLog$^{\text{vert}}$ (c) LinLog$^{\text{endv}}$

Figure 8.2: Improvement of layouts for the American-European Trade graph



(a) Normalized$^{\text{vert}}$ atedge length or total atedge length for ordering$^{\text{vert}}$

(b) Normalized$^{\text{endv}}$ atedge length or total atedge length for ordering$^{\text{endv}}$

Figure 8.3: Improvement of orderings for the American-European Trade graph

The following benefits and limitations are worth noting:

- For layouts, the LinLog energy model (as alternative formulation of the 0-normalized atedge length) is the first force/energy model that is not strongly biased towards uniform atedge lengths or uniform atvertex distances, and enables layouts that clearly reflect density (Figure 8.2a vs. Figure 8.2c, and Section 6.5). For layouts of graphs without vertex weights, the LinLog$^{\text{endv}}$ energy model is the first force/energy model that is not biased towards placing high-degree vertices at central positions, and enables layouts with improved external validity for some graphs with nonuniform degrees (Figure 8.2b vs. Figure 8.2c, and Section 6.5).
- For orderings of graphs without vertex weights, the normalized$^{\text{endv}}$ atedge length is the first quality measure that is not biased towards placing high-degree vertices at central positions. It enables orderings with improved external validity for some graphs with nonuniform degrees (Figure 8.3a vs. Figure 8.3b, and Section 5.3).
- For clusterings of graphs without vertex weights, the normalized$^{\text{vert}}$ and normalized$^{\text{endv}}$ atedge length are not the first unbiased$^{\text{vert}}$ and unbiased$^{\text{endv}}$ quality measures. Their optimal clusterings may be less useful than those of existing unbiased measures, like Newman's modularity, because they generally have only two clusters (see Section 4.2). However, computing optima is not the only appli-

cation of clustering quality measures, and the normalized atedge length still has the benefit of a clear interpretation, as ratio of the inter-cluster density to the overall density of the graph.

- There is more than one way to derive unbiased quality measures. In particular, an absolute quality measure can be divided through its expected value for graphs with uniform density (as in the normalized atedge length), or it can be reduced by its expected value (as in the shifted atedge length). Only the first alternative has been examined extensively in this work.
- For clusterings, the derivation of quality measures as difference of an absolute measure and its expected value for random graphs with uniform expected density has been proposed independently by Newman [New06], and less explicitly in some earlier works (see Subsection 4.3.6).
- The absence of bias is not sufficient to guarantee that a quality measure is internally valid. For example, any constant function is not biased, and it is possible to construct measures whose behavior for uniform-density graphs differs completely from the behavior for other graphs. In the normalized atedge length, however, the numerator is the average atedge length, and thus rewards the grouping of densely connected vertices, and the denominator is the average atvertex distance, and thus rewards the separation of sparsely connected vertices.

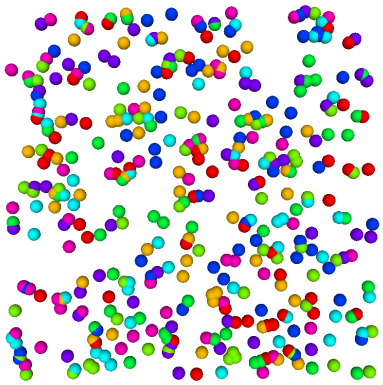
8.3 Simplification

Valid quality measures for graph assignments enable the automatic grouping of densely connected vertices with optimization algorithms. Besides the example assignments in Sections 4.4, 5.3, and 6.5 and in Part II, several successful applications have already been reported for the 0-normalized atedge length [vHvW04, vGVvdW04, MMB05, AOSB06][4] and for the shifted atedge length (see [New06] for references)[5]. In some examples, the detected structure is fairly subtle: Relative geographical distances between airports are discovered (though not with perfect precision) in an airline graph that does not contain any explicit information about geography, and semantically related terms are discovered in a hyperlink graph that does not contain any explicit information about semantics (see Section 6.5). And they are discovered with very simple means.

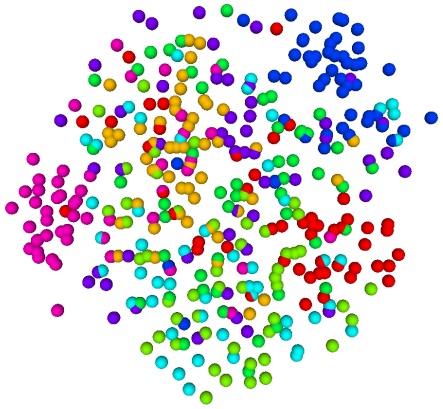
The optimization of a simple measure (the normalized atedge length with its variations) on simple system models (graphs) can suffice to detect nonobvious and useful structure in various real-world systems.

[4]The 0-normalized atedge length is an alternative formulation of the LinLog energy model for layouts (see Section 6.2), which was originally published by the present author in [Noa04].

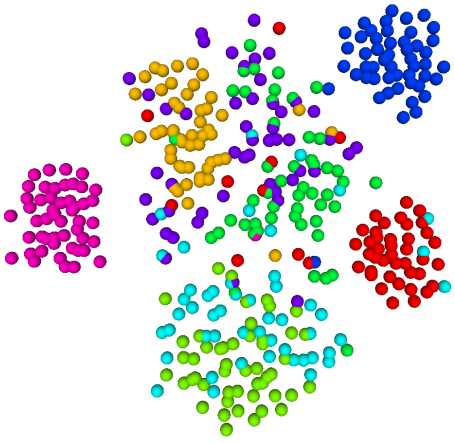
[5]The shifted atedge length is an alternative formulation of Newman's modularity measure for clusterings (see Section 4.3), which was originally introduced in [NG04, New04a].



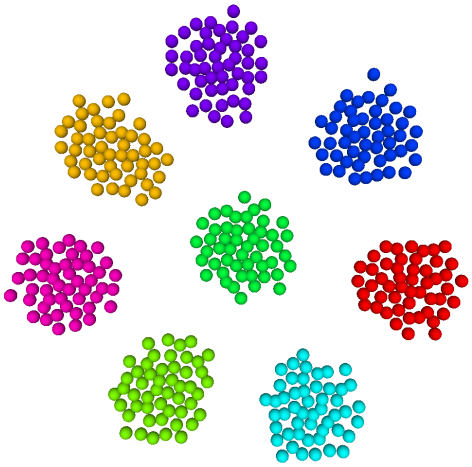
Normalized atedge length 1.000



Normalized atedge length 0.897



Normalized atedge length 0.791



Normalized atedge length 0.722

Figure 8.4: Detecting structure by optimizing the 0-normalized atedge length (illustrated for a pseudo-random graph with unit vertex weights, unit edge weights, an intra-cluster edge probability of 1.0, and an inter-cluster edge probability of 0.2)

Part II

Quality Criteria for Software Designs

Chapter 9

Introduction

9.1 Goal

Why is the physical world so comprehensible? Not least because of locality, notes physicist Paul Davies [Dav90]: If everything in the universe interacted with everything else in a highly non-local manner, we could never understand something without understanding everything. In large software systems, subsystem boundaries replace physical distances, but locality remains essential [Par72, SMC74]: Subsystems need to be comprehensible and changeable largely independently.[1] The goal of Part II are simple measures for the interdependence of software subsystems, that are general (though of course not exhaustive) indicators of software design quality.

To state this goal more precisely, some terms need to be defined. A software system is considered as a set of *software elements* that is partitioned into disjoint *subsystems*[2]. A subsystem s_1 is *dependent* on a subsystem s_2 if performing a development activity (like comprehension, modification, testing, or reuse) on s_1 requires performing a development activity on s_2. The *coupling* of two subsystems quantifies their degree of interdependence. The *total coupling* in a software system is the sum of the couplings of all pairs of different subsystems. The *cause* of a dependency is the part of a system whose modification or removal would remove the dependency. A dependency cause has a *leverage* of $\frac{k_1}{k_2}$ if it causes $k_1\%$ of the total coupling, and its size is $k_2\%$ of the total size of all dependency causes. If total coupling indicates design quality, then leverage indicates design flaws, i.e. small system parts that significantly impair design quality.

The three following chapters introduce three measures for the total coupling in a software system (each with one or two normalized variants) – called *total cochange coupling*, *total d-ref coupling*, and *total t-ref coupling* – and three corresponding mea-

[1] This is also true for artificial physical systems, as famously noted by Herbert Simon [Sim62] and Christopher Alexander [Ale64, Chapter 3].

[2] Depending on the system and the analysis goals, subsystems and software elements may be e.g. directories, packages, files, object classes, or functions.

sures of leverage. The coupling measures are essentially the total atedge cut and the total atpair cut of Part I, applied to specific graph models of software systems. They are based on two well-known and proven indicators of dependencies, namely cochange and references. Two software elements or subsystems are said to *cochange* if both are affected by a common logical change, e.g. a common bug fix or feature addition. A software element or subsystem s_1 *references* a software element or subsystem s_2 if s_1 uses an identifier that is declared in s_2. The goal of Part II is to provide evidence that the total cochange coupling, the total d-ref coupling, and the total t-ref coupling are general indicators of design quality.

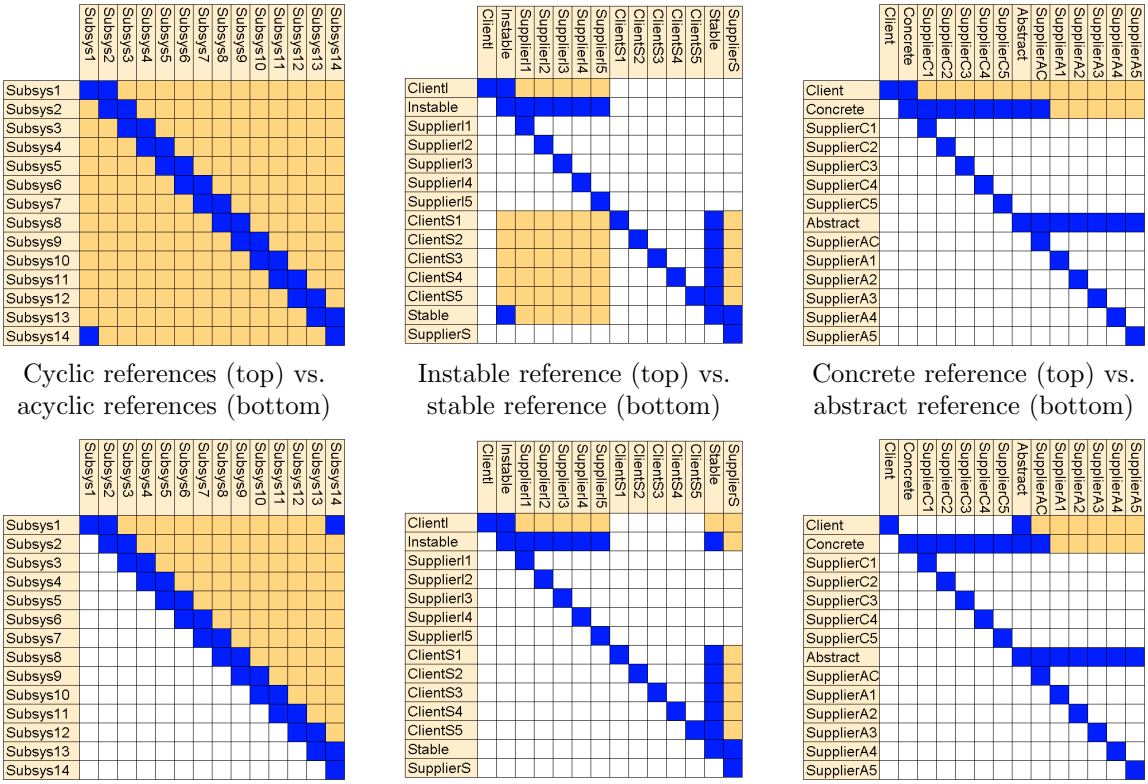
The following two subsections detail the two subgoals: the unification of existing design principles, design rules, design patterns, and design antipatterns, to obtain general indicators of design quality; and the validation of the coupling measures, to ensure that they indeed indicate design quality.

9.1.1 Unification

Goal The first subgoal of Part II is to demonstrate the generality of the total cochange coupling and the total t-ref coupling, by showing that they subsume several existing design principles, design rules, design patterns, and design antipatterns. (Generality will not be demonstrated for the total d-ref coupling, which is basically a simplified version of the total t-ref coupling.) A coupling measure is said to *subsume* a design principle, design rule, or design (anti)pattern, if improving the conformance to the principle or rule, introducing an instance of the pattern, or removing an instance of the antipattern significantly reduces the measurement value (all other things being equal). Here the term *design pattern* is used for recurring structures that generally improve design quality (similarly as in the like-named book [GHJV95]), and the term *design antipattern* is used for recurring structures that generally impair design quality (somewhat differently than in the like-named book [BMMM98]).

Two limitations should be noted. First, principles, rules, and patterns can be subsumed (in the above sense) by a coupling measure although they have other benefits besides reducing coupling; being subsumed does not mean becoming obsolete. Second, principles, rules and (anti)patterns are usually defined informally; thus no mathematical proofs, but only arguments for their subsumption can be provided; and not exactly principles, rules, and (anti)patterns, but rather specific interpretations can be subsumed.

Motivation Even without any generalization, the formalization of design principles and design rules to coupling measures enables the quantification and targeted optimization of the conformance to these principles and rules. Using a single general measure instead of many specific measures has the additional benefits that less measurement values need to be interpreted and optimized. And of course, simple, general, and formal indicators of design quality contribute to the understanding of what is good software design.



Cyclic references (top) vs. acyclic references (bottom)

Instable reference (top) vs. stable reference (bottom)

Concrete reference (top) vs. abstract reference (bottom)

Figure 9.1: Subsumption of three antipatterns by the total t-ref coupling

Example Figure 9.1 shows six matrix visualizations of software systems. Each row and each column represents a subsystem; a matrix element is blue if there is a reference from its row subsystem to its column subsystem; and a matrix element is orange if there is no direct reference but a path of references from its row subsystem to its column subsystem. The total t-ref coupling in a software system is represented by the non-white (i.e. blue or orange) area in its matrix visualization, ignoring matrix elements on the diagonal (i.e. self-references).

The three hypothetical software system in the upper row of Figure 9.1 contain instances of three different design antipatterns:

- a cycle of references (left);
- a reference from a stable subsystem, i.e. a subsystem with many ingoing and few outgoing references, to an instable subsystem, i.e. a subsystem with few ingoing and many outgoing references (middle); and
- a reference from a client not to an abstract interface subsystem, but to a concrete implementation of this interface (right).

The lower row of Figure 9.1 shows the same software systems as the upper row, except that the respective antipattern instances have been removed. In all three cases, the removal of the antipattern instance has significantly decreased the total t-ref coupling (non-white area). This illustrates that the total t-ref coupling subsumes the three antipatterns. More details can be found in Subsection 12.1.4.

Existing Results Several existing works relate design rules or design antipatterns to design measures (e.g. [EL96, LM06, SSM06]). However, these works apply one or several measures to each individual software element or subsystem, to detect violations of a *single* rule or instances of a single antipattern; in contrast, this work applies a single measure to the entire system, to quantify its conformance to *several* rules, and the absence of several antipatterns – and, as explained in the next subsection, to detect violations of all these rules, and instances of all these antipatterns.

Many coupling measures have been proposed in the literature (see [BDW99] for a survey), but the vast majority quantifies the coupling of a single software element to the remaining system, or the coupling between two software elements. Nevertheless, there are some measure that quantify the *total* coupling in a given system, and that are even similar to the special cases of the total cochange coupling, the total d-ref coupling, or the total t-ref coupling (or their normalized variants); see Subsections 10.2.5, 11.2.4, and 12.1.5 for details. However, non of these measures has been shown to subsume several design principles, rules, or (anti)patterns.

9.1.2 Validation

Goal The second subgoal of Part II is to provide evidence for the validity of the total cochange coupling, the total d-ref coupling, and the total t-ref coupling as indicators of design quality. A measure M is said to be an *indicator of design quality* if the following condition holds:[3]

- If a small change of a system (excluding changes of the subsystem decomposition), significantly decreases M, then it improves design quality.

Two equivalent restatements of this condition are:

- If the modification or removal of a small part p of a system significantly decreases M, then p is a design flaw (i.e. impairs design quality).
- System parts with large M-leverage are design flaws.

Indicators of design quality (in the defined sense) are clearly limited: They are not necessarily suitable for comparing different systems, for comparing entirely different designs of the same system, or for comparing different decompositions of the same set of software elements. Normalized versions particularly of the total cochange coupling might be suitable at least for the latter purpose, but this will not be evaluated.

Motivation If a measure of total coupling M is an indicator of design quality, then ordering all system parts by their M-leverage yields a prioritized list of design flaws. The process of computing this ordered list can be considered as incremental minimization of the measure M, because it searches for those system parts whose modification or removal decreases M most (relative to their size).

Of course, indicators of design quality can also be applied without systematic search and optimization, simply to compare different variations of a design, in order to choose the best variation.

[3] To be precise, the property of being an indicator of design quality is gradual rather than binary: The better a measure fulfils the condition, the more it is an indicator of design quality.

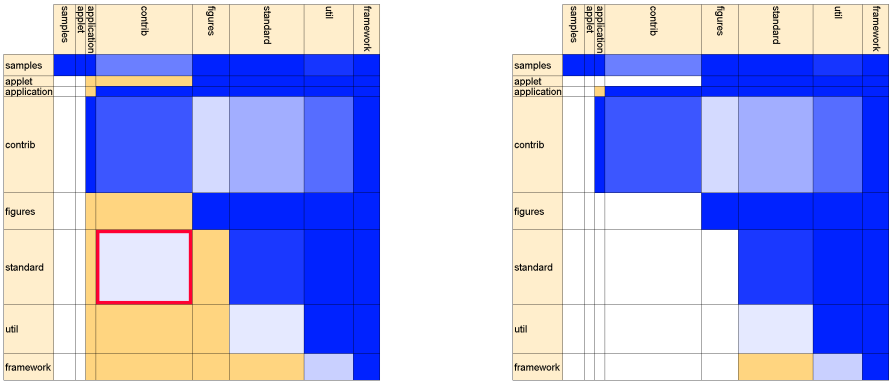


Figure 9.2: T-ref dependencies in JHotDraw 5.4

Example Figure 9.2 shows two matrix visualizations of the software system JHotDraw 5.4, which is described in Appendix A.3. The height of each row and the width of each column represents the size of the respective subsystem (in this case, its number of software elements); the color weight in each blue matrix element represents the size of the respective reference (in this case, the number of corresponding references between software elements). In the left matrix the reference from subsystem standard to subsystem contrib is marked, and in the right matrix it is removed. The size of this reference is only 0.36% of the total size of all inter-subsystem references, but its removal reduces the total t-ref coupling (off-diagonal non-white area) by about 35.1%, and thus its t-ref leverage of $35.1\%/0.36\% \approx 98$ is huge. This reference is indeed a design flaw, because the JHotDraw framework (except the subsystem samples) is intended to be reusable without the subsystem contrib.

Existing Results There is an extensive literature on the automatic detection of design flaws, in particular of rule violations (e.g. [EL96, Ciu99]) and of antipattern instances (e.g. [LM06, SSM06]). Several works address similar design flaws as the following chapters, namely strong change dependencies between subsystems [EW93, GJK03, BW03, DL06] and flawed references between subsystems (particularly [Mar03, Chapter 20]). The goal of this work is not primarily to introduce more precise techniques for the automatic detection of design flaws; the goal is to demonstrate that the optimization of simple coupling measures already suffices to detect various nontrivial design flaws, with similar and sometimes even better precision than existing techniques. This will be shown for the total cochange coupling, the total d-ref coupling, and the total t-ref coupling in Sections 10.3, 11.3, and 12.2.

Few previous works have attempted to detect design flaws by optimizing a single measure for an entire system [SBBP05, SSB06, OC06]. The measures used in these works are complex, and are not coupling measures; in fact, they do not measure any coherent property of software systems, but are arithmetic combinations of several measures for incomparable properties like size, complexity, cohesion, and coupling. Moreover, there is only very limited evidence that the computed results are actually design flaws.

9.2 Structure

The following chapters introduce and validate three measures for the total coupling in a software system, namely the total cochange coupling (Chapter 10), the total d-ref coupling (Chapter 11), and the total t-ref coupling (Chapter 12). The chapters can be understood largely independently, only Chapter 12 contains some references to Chapter 11; they build on the terminology from the beginning of this chapter.

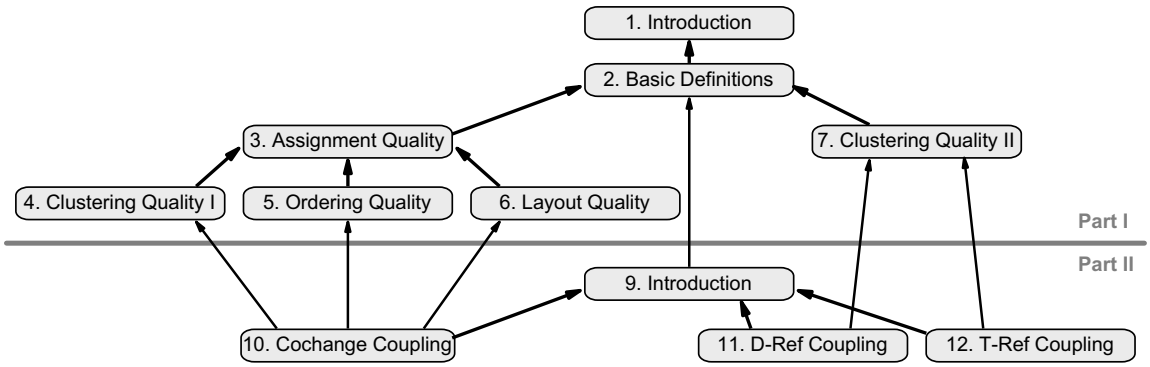


Figure 9.3: Comprehension dependencies between the chapters

The overall structure of this work is shown in Figure 9.3. Part II builds on Part I, although it is attempted to present at least the basic ideas in a self-contained manner. In particular, software systems with subsystem decompositions are modeled as graphs with clusterings (as defined in Chapter 2) and visualized as matrices and box-line diagrams (also defined in Chapter 2). The measures of cochange coupling in Chapter 10 are clustering quality measures from Chapter 4, and subsystems with large cochange leverage are grouped by optimizing ordering and layout quality measures from Chapters 5 and 6. The measures of d-ref coupling and t-ref coupling in Chapters 11 and 12 are clustering quality measures from Chapter 7.

9.3 Context and Limitations

The purpose of this section is to clarify the borderline and some relationships between contents and non-contents of Part II. The three subsections discuss the detection of dependencies (as parts of design models), the evaluation of design quality, and the improvement of design quality.

9.3.1 Detection of Dependencies

Dependencies between software elements are often not explicit in the source code, but need to be derived from certain indicators. This work contributes indicators for design quality and for design flaws which are based on two well-known indicators for dependencies, namely common changes and references. This work does *not* con-

tribute any new indicators for dependencies, or new graph models and extraction techniques for dependency indicators.

Common changes and references have been widely studied, and are believed to indicate a large number of dependencies with fairly high precision. The following paragraphs enumerate some further indicators of dependencies, to put common changes and references into perspective, and to point out further potential applications of the graph analysis techniques from Part I. The reader may safely skip these paragraphs, later chapters will not refer to their contents.

Control Flow There is a *direct control flow* from a statement s_1 to a statement s_2 if s_2 may be executed directly after s_1. *Control flow* is the transitive closure of direct control flow.

The comprehension of a method invocation statement s potentially requires the comprehension of all statements to which control may flow from s until it returns, because the effect of s is the cumulative effect of these statements. Also, the modification of a statement s potentially affects all statements that are executed after s, which justifies the use of control flow as an indicator of change dependencies in change impact analysis (e.g. [LR03, RST+04, OAH03, AOH05]).

In object-oriented programs, the direct control flows may differ significantly from the direct references [WH92]. However, in statically typed languages like Java or C++, the possible target classes of a direct control flow are restricted to the referenced class and its subclasses. For example, the object `b` in `void ma(B b) { b.mb(); }` must be of class `B` or one of its subclasses. The so-called Liskov substitution principle requires that subclasses conform to their superclass, more precisely, that their methods have the same or weaker preconditions and the same or stronger postconditions than the corresponding methods of the superclass ([Lis88], [Mey97, Section 16.1]). If this design principle is followed, then it suffices to understand the referenced method (in the above example, the method `mb` of the class `B`) to understand the method invocation, and arguably (invocation) references are a better indicator of comprehension dependencies than control flow.

Data Flow A *direct data flow* from a statement s_1 to a statement s_2 exists if s_1 assigns a value to a variable and s_2 may read this value before another assignment changes it. *Data flow* is the transitive closure of direct data flow. Direct data flow is one of the main edge types in the so-called program dependence graph [OO84] and system dependence graph [HRB90], where the term "dependence" means basically data and control dependence, and thus has a different (but related) meaning than in the present work. The statements with a data flow to a statement s are also called the *backward slice* of s, and the statements reached by a data flow from s are called the *forward slice* of s. Program slicing was introduced by Weiser [Wei81], and is surveyed in articles of Tip [Tip95] and Xu et al. [XQZ+05].

Changes may propagate along the data flow; if a statement is changed such that it assigns different values to a variable, then statements that read these values may also

need to be changed. Data flow has been used as indicator of change dependencies e.g. in [YCM78, HB85, GL91, CR00, Ton03, OAH03, BH05].

In the comprehension of a statement, it may be useful to focus on those parts of the system that affect the values of the variables at this statement. This holds in particular for the location of faults during debugging, which is one of the main applications of program slicing (e.g. [Wei82, WL86, ADS93]). If a variable has a wrong value at a statement s, then the backward slice of s contains the fault causing this wrong value (except if the fault consists of inadvertently missing statements). The forward slice of a statement s is also useful for debugging, because it shows how a value computed by s is used subsequently, and helps to ensure the value fulfils the assumptions of these later uses.

Data flow may also indicate reuse dependencies. The computation of a certain output can be extracted into a reusable function by determining the backward slice of the respective output statements (e.g. [NEK94, CLM96, LV97]). However, some intermediate results produced in this backward slice may be chosen to be input parameters of the reused function, and thus only parts of the slice may be reused.

Duplicates and Near-Duplicates Duplicated code (also called cloned code) induces change dependencies if several copies need to be kept consistent. The same problem concerns duplicates of design-level structures, like parallel inheritance hierarchies [Fow00, Chapter 3]. While a large number of methods for detecting code duplicates or near-duplicates has been proposed (as discussed in [DNR06, KG06, BKA+07]), relatively few works address the detection of higher-level clones (e.g. [MM01b, BJ05]).

Same Concept, Feature, Concern, or Aspect A *concept* is an abstract idea from the problem domain (like "payment" or "credit card") or the solution domain (like "list" or "sorting algorithm") of a development project [BMW94, MRB+05, PGM+07]. Because change requests are formulated in terms of concepts, software elements that implement the same concept are often changed or reused together in response to the same change request. A *feature* is a special concept describing an observable behavior of a system that can be triggered by the user [EKS03].

A *concern* is an issue that is addressed by a design decision, like the implementation of a feature, the representation of some data, or a mechanism for concurrency, distribution, persistence, failure recovery, etc. Code related to a concern changes whenever the respective design decision is changed. An *aspect* is a concern that cross-cuts the decomposition of a software system, i.e. whose implementation is not cleanly encapsulated in a single subsystems [KLM+97].

Concepts (including features) and concerns (in particular aspects) are often not explicit in the source code of a software system. Therefore, many methods for concept and feature location (surveyed in [WBP+03, MRB+05]) and aspect mining (surveyed in [CMM+05]) have been proposed. Most of these methods rely on the indicators of dependence that were listed in the previous paragraphs.

Similar Names and Comments As pointed out in the previous paragraph, software elements that implement similar concepts tend to be changed and reused together. Implemented concepts (or "the semantics", as some authors call it) can be identified by analyzing the names of software elements (e.g. [Nei96, AL98, GKS99, ZZL+06]), or identifier names and comments within the software elements (e.g. [AL99, MM01a, MSRM04, KDG05, PGM+07]).

Similar Authors or Same Owner If software elements need to be comprehended together and are expected to change together, then they are often assigned to the same developer or to the same owner, to enable changes to be performed or approved by a single person. Moreover, developers tend to introduce new dependencies rather to their own software elements than to software elements of other developers, because they have a better understanding and more control of these software elements.

The authors or owners of software elements can be extracted from source control logs, credits files, or copyright and change notices [BH99].

9.3.2 Evaluation of Design Quality

Quality is the totality of characteristics of an entity that bear on its ability to satisfy stated or implied needs [ISO94]. Software quality is commonly divided into *software product quality* and *software process quality*; the latter contributes to the former, but is not addressed in this work.

Taxonomies or models of software product quality like the standard ISO 9126 [ISO01] define *quality characteristics*, decompose them into subcharacteristics, and relate them to measurable *attributes*. For example, the quality characteristic of evolvability may be divided into subcharacteristics like comprehensibility, changeability, and testability. The interdependence of subsystems, as measured e.g. by the total d-ref coupling, is an attribute of software systems that affects these subcharacteristics. In this work, the term *design quality* is used to denote evolvability viewed at a medium or large scale, i.e. abstracting from the impact of implementation details.

Software product quality has an *internal* and an *external* aspect [ISO01]. Internal attributes of a product are determined purely by the product itself; external attributes also depend on the environment of the product, and are actually attributes e.g. of executions or modifications of the product. For example, the interdependence of subsystems is an internal attribute of a software product[4], while the cost of performing a specified set of modification tasks is an external attribute. In contrast to the interdependence of subsystems, the costs of modifications can only be directly measured when the modifications are performed and the costs actually occur, and cannot be directly optimized by changing the software system. Therefore, this work directly measures and optimizes the interdependence of subsystems, as an internal attribute, and provides evidence that it affects evolution costs, as the external attribute of primary interest.

[4] irrespective of the possible use of process data (e.g. historical changes) to measure this attribute

9.3.3 Improvement of Design Quality

Restructuring is the transformation of a software system from one representation form to another at the same relative abstraction level, while preserving the subject system's external behavior [CI90]. *Refactoring* is the process of changing a software system in such a way that it does not alter the external behavior of the code yet improves its internal structure [Fow00, p. xvi]. Refactoring is often considered as the object-oriented variant of restructuring (e.g. [MT04]). It is the main subject of the pioneering dissertation of Opdyke [Opd92], a widely known book of Fowler [Fow00], further books (e.g. [DDN02, Ker05, RL06]), and a research survey [MT04].

Parts of a software system that have a negative impact on certain quality characteristics are called *bad smells* in the refactoring community and *design flaws* in this work. The identification of design flaws is the only step of the refactoring process to which this work contributes. Specifically, it suggests that small system parts are likely design flaws if they strongly increase certain coupling measures, and introduces automatic and visual techniques for the identification of these system parts. Behavior-preserving transformations (also called refactorings) for reducing coupling are already described extensively in the literature; references will be provided, but new refactorings will not be introduced.

The design and implementation of *tools* for the detection of design flaws is outside the scope of this work.[5] Accordingly, the numerous existing tools will not be surveyed exhaustively, but will be mentioned if they provide unique dependency-based indicators of design flaws.

9.4 Summary

Goals:

- simple and general indicators of design quality and design flaws, based on (existing) graph models of common changes and references in software systems

Non-goals:

- new indicators, extraction techniques, and graph models for dependencies
- new transformations for the removal of design flaws (refactorings)
- processes and tools for quality assessment and improvement

[5] Actually, a tool was developed for the present work, but it is not considered as a primary result, and is therefore only outlined in Appendix B.

Chapter 10

Cochange Coupling and Cochange Leverage

In the development history of the software system ArgoUML 0.22, classes in the packages uml and ui have changed together 2226 times[1], much more often than the classes in any other pair of packages – because the packages uml and ui contain much more classes than any other pair of packages. Thus counting common changes provides insights about package size; more interesting insights require more sophisticated measures.

This chapter introduces a measure of coupling (as indicator of design quality) and a measure of leverage (as indicator of design flaws) for cochange dependencies, i.e. for dependencies that are indicated by common changes of different subsystems. The first section introduces the underlying model of common changes. The second section defines a measure of cochange coupling, and argues that it is related to the development costs caused by cochange dependencies. The third section derives a measure of cochange leverage, and examines empirically whether system parts with large cochange leverage are design flaws. The removal of the detected design flaws is not addressed in this chapter; existing catalogs [Fow00, Ker05, RL06] already contain many refactorings that aim at encapsulating changes.

10.1 Cochange

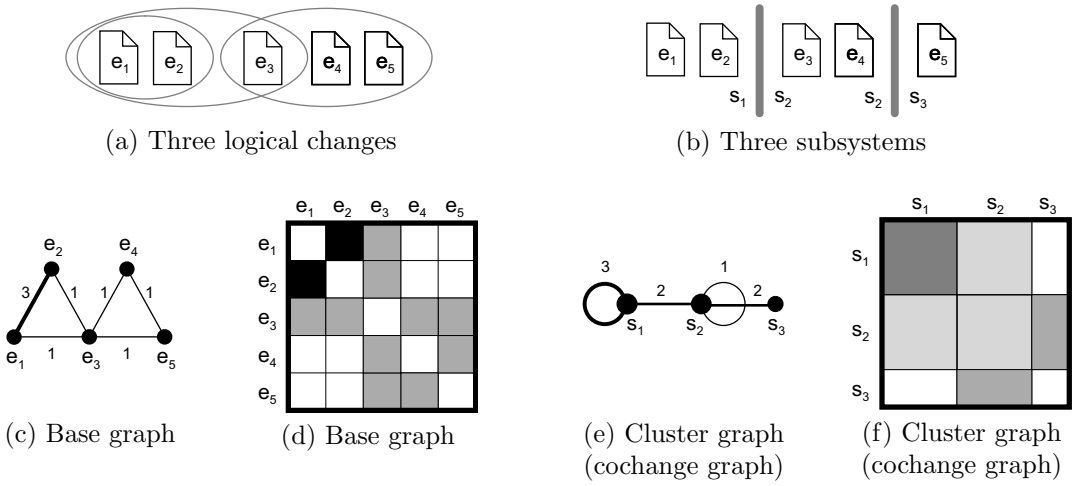
This section defines cochange and a graph model of cochange, and discusses related definitions from the literature.

10.1.1 Definition

A *logical change* comprises all modifications of a software system for a single coherent task, e.g. for a feature addition or a bug fix. An *individual change* is a modification

[1]More precisely, the cochange count of uml and ui is 2226. The cochange count is defined in Subsection 10.1.3. The extraction of the common changes is described in Appendix A.2.

of a single software element, as part of a logical change. The number of software elements that are modified in a logical change c, or equivalently the number of individual changes in c, is called the *size* of c and denoted as $|c|$. Logical changes of size 1 are ignored in this chapter because the focus is on how software elements change *together*. A logical change is called *nonlocal* if it affects software elements in more than one subsystem, and *local* otherwise. For example, the rightmost logical change in Figure 10.1a affects three software elements (e_3, e_4, and e_5) and two subsystems (s_2 and s_3, see Figure 10.1b), and thus has size 3 and is nonlocal.



(a) Three logical changes (b) Three subsystems

(c) Base graph (d) Base graph (e) Cluster graph (cochange graph) (f) Cluster graph (cochange graph)

Figure 10.1: Cochange models for five software elements

Cochange (common change) is defined for a given set of logical changes C. The *cochange strength* of each unordered pair of software elements $\{e_1, e_2\}$ is defined as $\sum_{c \in C : \{e_1, e_2\} \subseteq c} \frac{2}{|c|-1}$. That is, each modification of e_1 and e_2 in a common logical change c increases their cochange strength by a value that is roughly inversely proportional to the size of c. The cochange strength of each unordered pair of subsystems is the sum of the cochange strengths of the corresponding unordered pairs of software elements. Two software elements or subsystems with nonzero cochange strength are said to *cochange*. The cochange strengths resulting from the logical changes in Figure 10.1a are shown in Figures 10.1c (for software elements) and 10.1e (for subsystems).

Observe that each logical change c contributes to the cochange strength of $\frac{1}{2}|c|(|c|-1)$ unordered pairs of software elements; thus the total contribution of c to the cochange strength is $|c|$, i.e. its size; and the total cochange strength contributed by all logical changes is their total size, or equivalently the total number of individual changes. Moreover, a logical change c contributes to the cochange strength of each of its modified software elements with $|c|-1$ other software elements; thus the contribution of c to the total cochange strength (in graph terms, to the degree) of each of its modified software elements is 2; and the total cochange strength (i.e. the degree) of each software element is twice the number of its modifications.

A mapping of software systems with a subsystem decomposition and a set of logical changes to graphs with a clustering is defined in the upper part of Table 10.1. The resulting cluster graph is called the *cochange graph* of a software system. To simplify the presentation, the size of each software element is assumed to be 1 in this chapter, and thus (by the definition of cluster graph) the size of each subsystem is the number of its software elements. The actual construction of a cochange graph requires specific definitions of the terms logical change (or coherent task), software element, and subsystem; example definitions can be found in Appendix A.2.

Software System	Graph
software element	base vertex
size of software element (here always 1)	weight of base vertex
element cochange	base edge
strength of element cochange	weight of base edge
subsystem	cluster
subsystem size	weight of cluster
subsystem cochange	cluster edge
strength of subsystem cochange	weight of cluster edge
cochange dependency of two subsystems	cluster edge
cochange coupling of two subsystems	weight of cluster edge
total cochange coupling	total atedge cut of the cluster graph
scaled cochange coupling	scaled atedge cut of the cluster graph
normalized cochange coupling	normalized atedge cut of the cluster graph
cochange cause	connected pair of clusters
size of cochange cause	product of cluster weights
leveraged cochange cause	densely connected pair of clusters

Table 10.1: Graph model of cochange (upper part), cochange coupling (central part), and cochange leverage (lower part)

10.1.2 Visualization

Like other graphs, cochange graphs can be visually represented as box-line diagrams and as matrices (see Section 2.4), as shown in Figure 10.1e and 10.1f. In box-line visualizations, the subsystems are represented by circles, the subsystem sizes by the areas of the circles, and the cochange strengths by the widths of lines. (In Figure 10.1, the cochange strengths are additionally specified numerically.) In matrix visualizations, the subsystems are represented by rows and columns, the subsystem sizes by the row heights and columns widths, and the cochange strengths by the color weights in the matrix elements.

In the visualizations of real-world software systems, in particular on pages 171 to 175, slight deviations from these rules improve the readability. In the box-line diagrams, the lines are elided; the cochange strengths are still indicated by the layouts, as explained in Section 10.3. In the matrices, small rows and columns are enlarged to ensure the readability of the subsystem names, and the maximum possible color density represents all cochange densities above a certain large threshold.

10.1.3 Related Work

Models In most previous models of common change, the edge weight between each unordered pair of software elements $\{s_1, s_2\}$ is their *cochange count*, i.e. the number of logical changes that affect both s_1 and s_2 [EW93, MW01, GJK03, ZDZ03, BAY03, BW03, YMNCC04]. With this definition, each logical change c contributes 1 to the weight of $\frac{1}{2}|c|(|c|-1)$ edges; thus the total contribution of c to the edge weights is roughly proportional to the *square* of its size. This disproportionally high weighting of large logical changes is not justified in the mentioned works.

If each logical change has the same size k, then the cochange strength differs from the cochange count only by a constant factor of $\frac{2}{k-1}$, for each pair of software elements or subsystems. Even if the sizes of the logical changes vary, the cochange strength and the cochange count are still roughly proportional if they are large, because the relative variation of the *average* size of *many* logical changes tends to be smaller than the relative variation of the sizes of individual logical changes. This is illustrated in Figure 10.2 for all unordered subsystem pairs of two example software systems (which are described in Appending A.2).

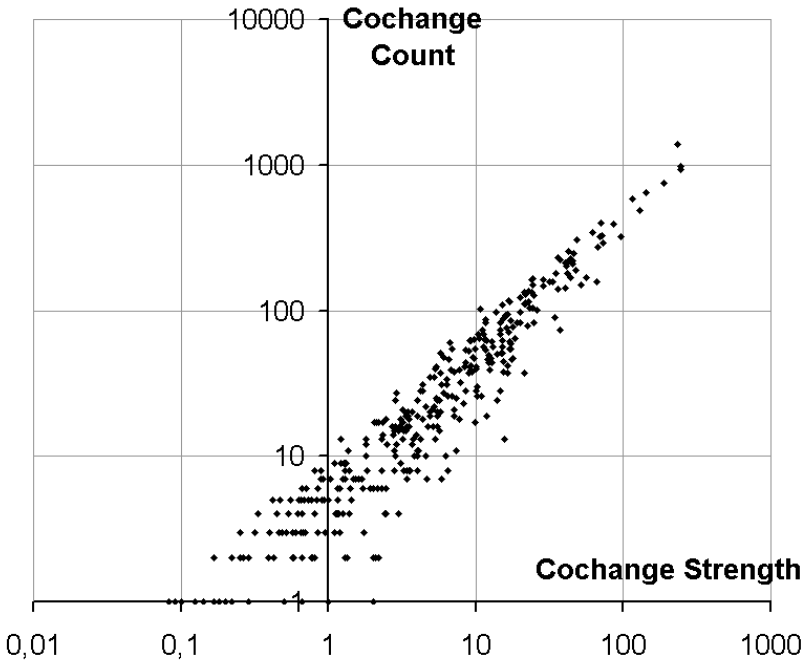
In a joint publication with Beyer [BN05], the present author introduced a graph model of common change where both software elements and logical changes are represented as vertices, and each logical change is connected to each of its modified software elements by an edge. In the present work, explicit vertices for logical changes are not needed and thus omitted.

Some models for the common change of software elements are not based on logical changes, but e.g. on similar change times [RD04, ARV05, BGA06], or changes (or similar changes) in similar versions of the software system [GHJ98, XS06, GDK+07]. A detailed description and discussion of these models is beyond the scope of this work; models based on logical changes are currently more widely used and more proven.

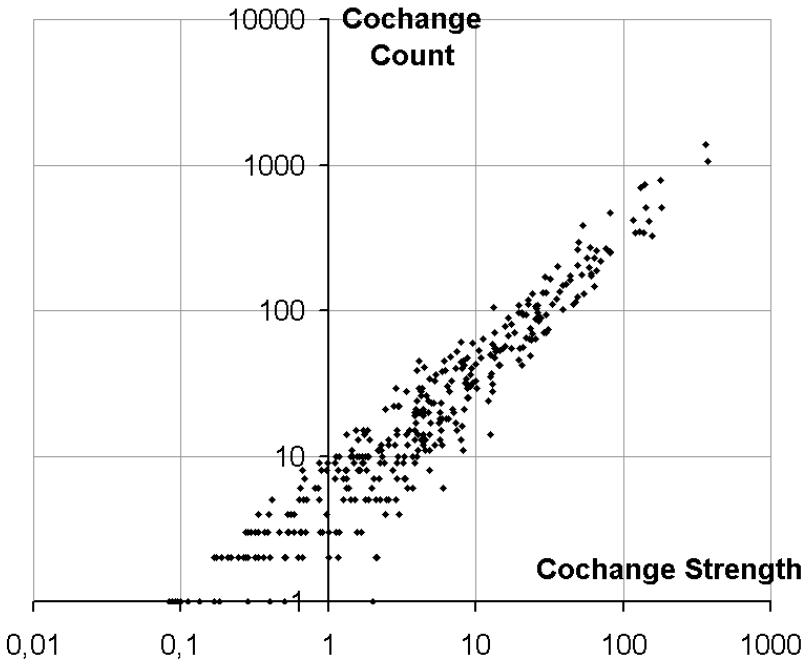
Models that combine common change with other indicators of dependencies (e.g. [BW03, CMSB05, FG06a]) potentially enable more precise and more extensive analyses. Nevertheless, there is value in examining pure cochange models, because a better understanding of these basic models also improves the understanding of their extensions, and because they may be easier to extract and to interpret in practice.

Visualizations Many works on common change include box-line visualizations of their cochange models; matrix visualizations are less common [ZDZ03, BDW05]. The matrix visualizations in this work use a specific mapping of graph properties like cochange strength and cochange density to visual properties like color weight and color density, to support the visual assessment of cochange coupling and cochange leverage (discussed in the next sections).

The visualizations of D'Ambros and Lanza [DL06] and Xie et al. [XPM06] are less related, because they represent the cochange of a single software element or subsystem with the other software elements of subsystems, and not the cochange of all pairs of software elements or subsystems.



(a) ArgoUML 0.22



(b) Eclipse 3.1, plug-in jdt.ui

Figure 10.2: Cochange strength vs. cochange count in two software systems, for all unordered pairs of subsystems with nonzero cochange strength

10.1.4 Past Cochange vs. Future Cochange

The cochange graph was defined with respect to a set of logical changes, but it was left open which set of logical changes should be used, because this depends on the analysis goals. The goal of this chapter are indicators for changeability, i.e. for the capability of software systems to enable future changes. Therefore, it will be assumed that the cochange graph reflects *future* logical changes, unless stated otherwise.

To apply the introduced indicators of changeability in practice, a prediction of future logical changes is required. The most sophisticated variant of the proposed coupling measure (the normalized cochange coupling) and the proposed measure of cochange leverage depend only on the *relative* cochange strengths of the software elements; thus it suffices to predict the cochange strengths up to a constant factor.

The development and evaluation of methods for predicting relative cochange strengths is not a goal of this work. The literature suggests that the relative cochange strengths in the future are often similar to the relative cochange strengths in the past. Several works provide direct empirical evidence for this similarity [YMNCC04, HH04a, ZWDZ05, Ste05], and further works ([EW93, GJK03, ZDZ03, DL06], Section 10.3 of this chapter) provide indirect evidence by showing that useful knowledge about the current design quality can be derived from past common changes.

10.2 Cochange Coupling

This section introduces a measure of cochange coupling (with three variants), shows how its values are represented in visualizations of the cochange graph, and discusses related measures. The third and fourth subsection provide evidence for the validity and generality of the measure as indicator of design quality, by establishing a relation to the additional development costs caused by the nonlocality of changes, and by showing that minimizing the measure subsumes widely accepted design principles, design rules, and design patterns.

10.2.1 Definition

Two different subsystems are *cochange dependent* if they cochange. The *cochange coupling* between two different subsystems is the sum of the cochange strengths between their software elements.[2] The *total cochange coupling* in a software system is the sum of the cochange strengths between software elements in different subsystems, or equivalently the total atedge cut of the cochange graph.[3] The *scaled cochange coupling* is the average cochange strength between software elements in different subsystems, or equivalently the scaled atedge cut of the cochange graph.

[2]Thus the cochange coupling between two subsystems equals their cochange strength. Cochange coupling and cochange strength are nevertheless distinguished, to separate the model of cochange from measures based on this model.

[3]The total atedge cut, the scaled atedge cut, and the normalized atedge cut are clustering quality measures that were defined in Chapter 4.

Unlike the total cochange coupling, it is not biased towards coarse-grained subsystem decompositions. The *normalized cochange coupling* is the average cochange strength between software elements in different subsystems (i.e. the scaled cochange coupling) divided by the average cochange strength between any two software elements, or equivalently the normalized atedge cut of the cochange graph. Unlike the scaled cochange coupling, it is not biased towards a small total size of the logical changes. Note that only nonlocal changes, i.e. logical changes that affect more than one subsystem, increase the cochange coupling.

In Figure 10.1 on page 152, the cochange coupling between the subsystems s_1 and s_2 is 2, the total cochange coupling is 4, the scaled cochange coupling is $\frac{4}{8} = \frac{1}{2}$, and the normalized cochange coupling is $\frac{4}{8}/\frac{8}{10} = \frac{5}{8}$.

A normalized cochange coupling smaller than 1 indicates that the cochange between software elements in different subsystems is weaker (on average) than between software elements in the same subsystem. It may be hypothesized that in most reasonably large software systems, the normalized cochange coupling is significantly smaller than 1, because otherwise their controlled evolution would be impossible. An experimental verification of this hypothesis is beyond the scope of this work, but at least it is not disproved by the real-world examples in this chapter: For the three decompositions of the software systems ArgoUML and Eclipse (plug-in jdt.ui) shown on pages 171 to 175 and described in Appendix A.2, the normalized cochange coupling based on *past* logical changes is 0.574, 0.451, and 0.428, respectively.

10.2.2 Visualization

The total and the scaled cochange coupling in a software system are directly reflected in the matrix visualization of the cochange graph, as detailed in Chapter 4 for the corresponding clustering quality measures. The cochange coupling between two different subsystems is the color weight in each of the two corresponding matrix elements. The total cochange coupling equals the total color weight above the diagonal elements (or equivalently below the diagonal elements), and the scaled cochange coupling equals the average color density outside the diagonal elements.

In Figure 10.1f on page 152, the cochange coupling between the subsystems s_1 and s_2 equals the cochange coupling between s_2 and s_3, and accordingly the color weight in the corresponding matrix elements is equal. (Comparing the color density of matrix elements is easier than comparing the color weight, and is in fact more important, as discussed in the next section.) Removing the cochange coupling between s_1 and s_2 halves the total cochange coupling from 4 to 2 and the scaled cochange coupling from $\frac{1}{2}$ to $\frac{1}{4}$, and accordingly halves the total color weight and the average color density outside the diagonal.

10.2.3 Relation to Development Costs

This subsection shows that under two simplifying assumptions, the total cochange coupling for a set of logical changes C is proportional to the total cochange costs

of C. The *cochange costs* of a nonlocal change c are the additional costs for performing c that occur because c is nonlocal, i.e. affects more than one subsystem; the cochange costs of a local change are 0. The nonlocality of a logical change generally causes additional costs, because additional subsystems need to be comprehended and retested, and often additional developers or even development sites need to be involved. However, there are also logical changes that do not benefit from locality, for example modifications of the copyright notice or of the indentation; these logical changes should be excluded if the locality of changes is analyzed with the goal of assessing or improving changeability.

The two assumptions used to derive the total cochange coupling are very simple and are not claimed to hold exactly in practice; accordingly, the total cochange coupling and the cochange cost are not claimed to be exactly proportional in practice. The goal is rather to motivate the precise definition of the total cochange coupling, and to provide evidence for a relation between the total cochange coupling and changeability. The assumptions are:

1. The cochange costs of a logical change c are proportional to its size $|c|$. The definition of the size of a logical change as the number of affected software elements in Section 10.1 is only a simple example (which already enables practically useful results, as shown in Section 10.3); the assumption may be a better approximation of reality for other definitions of size, e.g. as effort in person hours.

2. The cochange costs of a logical change c are proportional to its *degree of nonlocality*, which is defined as the fraction of inter-subsystem pairs of software elements among all unordered pairs of modified software elements, or formally as $\sum_{\{e_1,e_2\}\subseteq c\,:\,\mathrm{p}(e_1)\neq\mathrm{p}(e_2)} 1 \;/\; \sum_{\{e_1,e_2\}\subseteq c} 1$ (where $\mathrm{p}(e)$ is the subsystem of the software element e). The degree of nonlocality takes the minimum value 0 if each modified software element belongs to the same subsystem (i.e. if the logical change is local), and takes the maximum value 1 if each modified software element belongs to a different subsystem.

With these assumptions, the cochange costs of a logical change c for a subsystem decomposition p are proportional to the product of its size and its nonlocality, i.e. to

$$|c| \cdot \frac{\sum_{\{e_1,e_2\}\subseteq c\,:\,\mathrm{p}(e_1)\neq\mathrm{p}(e_2)} 1}{\sum_{\{e_1,e_2\}\subseteq c} 1} \;=\; \frac{2}{|c|-1} \cdot \sum_{\{e_1,e_2\}\subseteq c\,:\,\mathrm{p}(e_1)\neq\mathrm{p}(e_2)} 1 \;.$$

For a set of logical changes C, the total cochange costs are proportional to

$$\sum_{c\in C} \frac{2}{|c|-1} \cdot \sum_{\{e_1,e_2\}\subseteq c\,:\,\mathrm{p}(e_1)\neq\mathrm{p}(e_2)} 1 \;.$$

If E is the set of software elements in the software system, this equals

$$\sum_{\{e_1,e_2\}\in E^{(2)}\,:\,\mathrm{p}(v_1)\neq\mathrm{p}(v_2)} \sum_{c\in C\,:\,\{e_1,e_2\}\subseteq c} \frac{2}{|c|-1} \;.$$

The inner sum is the cochange strength of $\{e_1, e_2\}$, and thus the entire term equals the total atedge cut of the cochange graph.

10.2.4 Relation to Design Principles and (Anti)Patterns

The total cochange coupling grows with the number and size of nonlocal logical changes, and, under simplifying assumptions, even with the costs caused by nonlocality. Thus the minimization of total cochange coupling is one particular formalization (of many possible formalizations) for the design principle of *locality of change*:

> Logical changes should affect only one subsystem.

Locality of change is also denoted as *modularity*; for example, the IEEE Standard Glossary of Software Engineering Terminology defines modularity as [IEE90]:

> The degree to which a system or computer program is composed of discrete components such that a change to one component has minimal impact on other components.

The remainder of this subsection lists design principles, rules, patterns, and antipatterns that are subsumed by locality of change, in the sense defined in Subsection 9.1.1. Note that the quotations use different terms to denote subsystems, including module, package, and class.

Locality of change subsumes information hiding, which Parnas summarizes as follows [Par72, p. 1058]:

> We propose instead that one begins with a list of difficult design decisions or design decisions which are likely to change. Each module is then designed to hide such a decision from the others.

Locality of change corresponds to the second part ("... and no other package") of Martin's Common-Closure Principle [Mar03, Chapter 20]:

> The classes in a package should be closed together against the same kinds of changes. A change that affects a package affects all classes in that package and no other packages.

Many object-oriented design heuristics of Riel [Rie96] are primarily motivated by locality of change, for example:

- All data should be hidden within its class (Heuristic 2.1).
- Keep related data and behavior in one place (2.9).
- Minimize the number of classes with which another class collaborates (4.1).
- Derived classes must have knowledge of their base class by definition, but base classes should not know anything about their derived classes (5.2).

Locality of change also subsumes many object-oriented design patterns, as noted by Gamma et al. [GHJV95, p. 24]:

> Each design pattern lets some aspect of system structure vary independently of other aspects, thereby making a system more robust to a particular kind of change.

Several antipatterns ("bad smells") of Beck and Fowler are violations of locality of change, and are explicitly introduced as such [Fow00, Chapter 3]: Duplicated Code, Shotgun Surgery, Feature Envy, Switch Statements, Parallel Inheritance Hierarchies, Message Chains, Inappropriate Intimacy.

10.2.5 Related Work

Most existing measures of cochange coupling quantify the coupling between two software elements or subsystems, and not the coupling of all subsystems in a software system. Such measures are potentially applicable for locating design flaws, and are therefore discussed in Subsection 10.3.4.

Zimmermann et al. propose two measures called *evolutionary density index EDI* and *evolutionary coupling index ECI*, to quantify the nonlocality of changes [ZDZ03]. In both measures, pairs of software elements are considered as either coupled or not coupled, depending on whether certain measures for their cochange exceed given thresholds. Compared to the total cochange coupling and its normalized variants, which factor in the precise values of the cochange strength, this binarization means a loss of information. The EDI is the ratio of the number of coupled pairs of software elements to the number of all pairs of software elements. It is thus similar to the special case of the scaled cochange coupling where each subsystem contains exactly one software element. The ECI is the ratio of the number of coupled inter-subsystem pairs to the number of coupled intra-subsystem pairs of software elements. It is biased towards coarse grained subsystems, as it is always 0 if all software elements belong to the same subsystem, and maximal if all software elements belong to different subsystems.

10.3 Cochange Leverage

This section derives a measure of leverage from the measure of cochange coupling defined in the previous section. It examines empirically whether this measure is a precise indicator of design flaws, and examines both theoretically and empirically whether it is a more precise indicator of design flaws than similar existing measures.

10.3.1 Definition

In this section, pairs of subsystems are exemplarily considered as causes of cochange coupling, and thus a measure of cochange leverage is derived and evaluated for subsystem pairs. Alternatives, namely pairs of software elements, individual software elements, and individual subsystems, are discussed at the end of this subsection.

As defined in Section 9.1, a dependency cause has a leverage of $\frac{k_1}{k_2}$ if it causes $k_1\%$ of the total coupling, and its size is $k_2\%$ of the total size of all dependency causes. The contribution of each unordered pair of subsystems to the total cochange coupling is its cochange strength, and the size of each unordered pair of subsystems may be defined as the product of the subsystem sizes (or equivalently as the number of the corresponding unordered pairs of software elements, if each software element has the size 1). Thus, for a cochange graph $((V, \mathrm{w}), (E, \mathrm{f}))$, the *cochange leverage* of a subsystem pair $\{s_1, s_2\} \in E$ is

$$\frac{\mathrm{f}(\{s_1, s_2\})}{\sum_{\{v_1,v_2\}\in E:\, v_1\neq v_2} \mathrm{f}(\{v_1, v_2\})} \Big/ \frac{\mathrm{w}(s_1)\mathrm{w}(s_2)}{\sum_{\{v_1,v_2\}\in E:\, v_1\neq v_2} \mathrm{w}(v_1)\mathrm{w}(v_2)} \,.$$

For a fixed cochange graph, the denominators of the two fractions (i.e. the total cochange coupling and the total size of the subsystem pairs) are constant, and the cochange leverage is proportional to $\frac{\mathrm{f}(\{s_1,s_2\})}{\mathrm{w}(s_1)\mathrm{w}(s_2)}$, i.e. to the inter-density of s_1 and s_2. Thus large cochange leverage means dense cochange.

Given the model of cochange from Subsection 10.1.1, there are four obvious choices for causes of cochange coupling: software elements (base vertices), pairs of software elements (base edges), subsystems (cluster vertices), and pairs of subsystems (cluster edges). Comparing these alternatives, dense cochange between two subsystems implies dense cochange between at least some of their software elements, but not vice versa: If only few pairs of software elements in two subsystems cochange densely and the remaining pairs cochange sparsely, then the two subsystems cochange rather sparsely. Similarly, dense cochange of an individual subsystem with the remaining system implies dense cochange of the subsystem with at least some other subsystems, but not vice versa. Thus the four alternatives are primarily different tradeoffs between detailedness and brevity. In practice, tools may allow the user to choose this tradeoff interactively.

10.3.2 Visualization

Matrix Visualizations and Orderings In matrix visualizations of the cochange graph, each subsystem pair is represented by two off-diagonal matrix elements. The cochange leverage of a subsystem pair is proportional to the inter-density of the subsystems, and thus to the color density of the two matrix elements. Thus subsystem pairs with large cochange leverage correspond to off-diagonal matrix elements with large color density.

Graph orderings with small normalized atedge length group densely connected vertices and separate sparsely connected vertices (see Chapters 3 and 5). Thus matrix visualizations of the cochange graph with such orderings have two additional benefits. First, subsystem pairs with large cochange leverage can be identified more easily, because the corresponding matrix elements tend to be close to the diagonal. Second, groups of more than two densely cochanging subsystems can be identified as squares with high color density along the diagonal. Obviously, detecting a group of k densely cochanging subsystems at a glance is much more efficient than detecting $\frac{1}{2}k(k-1)$ pairs of densely cochanging subsystems separately.

Figure 10.3b shows a matrix visualization of the cochange graph of the software system ArgoUML 0.22. The ordering is pseudo-random and has a normalized atedge length of 0.97. The marked matrix elements correspond to subsystem pairs with large leverage. Figure 10.4b on page 171 shows the same cochange graph with the same marked subsystem pairs, but a different ordering with a smaller normalized atedge length of 0.53. While the subsystem pairs with the largest leverage can already be identified in Figure 10.3b by their large color density, their identification is easier in Figure 10.4b because they are close to the diagonal. More importantly, only Figure 10.4b shows the existence of groups of several densely cochanging subsystems, most notably the group of several subpackages of uml.diagram in the center.

Box-line Visualizations and Layouts A cochanging subsystem pair is an edge of the cochange graph, and is thus represented by a line in box-line visualizations of the cochange graph. The cochange leverage of subsystem pairs is not explicitly reflected in box-line visualizations as introduced in Section 2.4. It could be reflected by setting the color density of each line to the inter-density of the corresponding subsystem pair, as illustrated in Figure 10.3a. However, the figure shows that subsystem pairs with large cochange leverage are still difficult to identify, because there are too many lines. In visualization tools like EvoLens [RFG05], the lines can be filtered interactively; in the static visualizations of this chapter (except Figure 10.3a), the lines are omitted entirely, and the densities are indicated by layouts.

Layouts of the cochange graph with small normalized atedge length group densely cochanging subsystems and separate sparsely cochanging subsystems (see Chapters 3 and 6). For example, Figure 10.4a on page 171 displays a layout with a normalized atedge length of only 0.48. This layout shows not only several pairs of densely cochanging subsystems, and in particular the densely cochanging group of subpackages of uml.diagram, but also that subsystems like uml.util, uml.notation and notation cochange sparsely with the other subsystems. However, there are also sparsely cochanging subsystems that are placed closely together, like application and ocl, and densely cochanging subsystems that are placed far apart, like kernel and util. This is generally unavoidable because the Euclidean distances in layouts are restricted by the triangle inequality and the limited number of dimensions. Tables (like Table 10.2 on page 170) show individual cochange densities more precisely than layouts, but are much less effective in showing densely cochanging groups; thus tables and layouts complement each other.

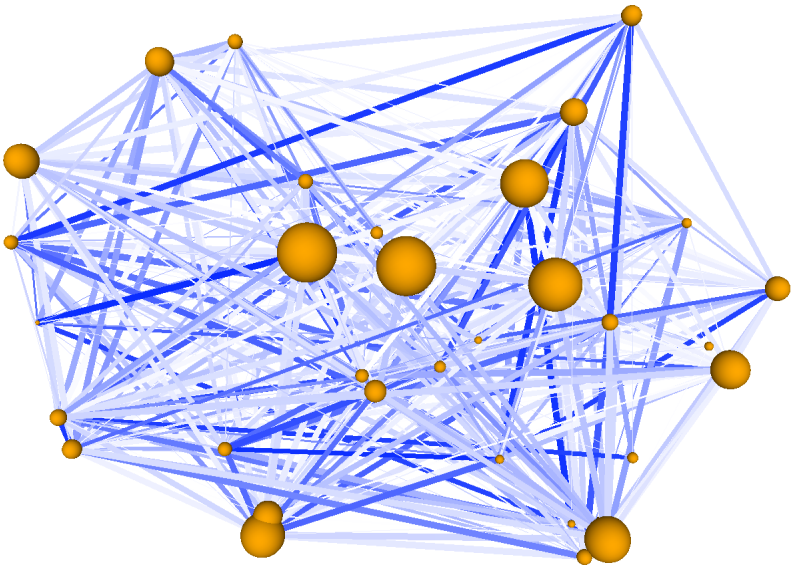
10.3.3 Relation to Design Flaws

An empirical study was performed to evaluate two hypotheses:

- Software systems contain system parts with a cochange leverage much greater than 1; in other words, small parts of the systems cause a large part of their total cochange coupling.
- A large fraction of the system parts with high cochange leverage are design flaws. (If the cochange leverage is an indicator of design flaws, then the total cochange coupling is an indicator of design quality, see Subsection 9.1.2).

Method Appendix A.2 describes the analyzed software systems ArgoUML 0.22 and Eclipse 3.1 (plug-in jdt.ui), and the extraction of their cochange graphs. Essentially, the past cochange strengths of the software elements have been computed from the version repositories of the systems, and have been used as predictions of the future cochange strengths.

To examine the existence of system parts with large cochange leverage, the cochange leverage of all subsystem pairs was computed, and the subsystem pairs of each system were ordered by decreasing cochange leverage.



(a) Box-line visualization with pseudo-random layout

(b) Matrix with pseudo-random ordering; the 29 most leveraged subsystem pairs are marked

Figure 10.3: Cochange dependencies in ArgoUML

To assess whether system parts with large cochange leverage are design flaws, the subsystem pairs of each system were examined in the order of decreasing leverage. Subsystem pairs with a cochange strength smaller than 10 were skipped, because small couplings are susceptible to noise in the change data. For each remaining subsystem pair with large cochange leverage, it was checked whether it can be narrowed to an even more leveraged cochange cause with almost the same cochange strength, by excluding pairs of software elements with very small cochange strength. The remaining pairs of software elements were classified as design flaws if they fulfilled at least one of the following criteria:

- Similar responsibilities; software elements with very similar responsibilities should generally not belong to different subsystems.
- Strong structural relationships, like inheritance from complex superclasses, invocation of an unusually large number of different methods, or cyclical relationships.
- Strong structural similarities, like common superclasses or duplicated code.

Results: Existence of Large Cochange Leverage Tables 10.2 and 10.4 on pages 170 and 174 show that both software systems contain many subsystem pairs with a cochange leverage significantly greater than 1. The maximum cochange leverage is 63.5 for ArgoUML and 17.4 for jdt.ui. Only 0.40% (ArgoUML) and 1.01% (jdt.ui) of the subsystem pairs induce 10% of the total cochange coupling.

Results for ArgoUML The subsystem pairs with the largest cochange leverage are listed on page 170; a matrix visualization and a box-line visualization of the cochange graph are shown on page 171. The ordering in the matrix visualization has been computed with a heuristic for minimizing the normalized atedge length (as in Section 5.3); the layout in the box-line visualization has been computed with a heuristic for minimizing the LinLog energy (as in Section 6.5). Thus subsystems with large cochange leverage are grouped in both visualizations.

The subsystems are packages; the software elements are source code files, and mostly coincide with top-level classes. For each subsystem pair, particular pairs of software elements are discussed; these are not hand-picked examples, but the pairs that have the largest cochange strength, and that together contribute most of the cochange strength of their subsystems.

{kernel, uml.diagram.diagram[4]} The classes Project and ProjectMember strongly cochange with the class ProjectMemberDiagram[5]. The class ProjectMember is a superclass of ProjectMemberDiagram, and Project and ProjectMemberDiagram

[4]The name uml.diagram.diagram is used for the classes that are *directly* contained in the package uml.diagram. In contrast, the name uml.diagram would be used for the classes in the package uml.diagram *and its subpackages*. This convention is used in all similar cases.

[5]For brevity, the classes are not qualified with the package names, but mentioned in the same order as their packages; i.e. Project and ProjectMember belong to kernel and ProjectMemberDiagram belongs to uml.diagram.diagram.

are cyclicly related through mutual imports. Thus the large cochange leverage indicates similar responsibilities and strong structural relationships.

{uml.diagram.collaboration, uml.diagram.deployment} The matrix and the box-line visualization show that not only this pair, but all subpackages of uml.diagram (except uml.diagram.layout) cochange densely. The packages uml.diagram.ui and uml.diagram.diagram contain the large classes UMLDiagram, FigNodeModelElement, FigEdgeModelElement, and UMLMutableGraphSupport (among others), which are subclassed in each of the other packages. A large part of the cochange strength is contributed by the cochange of these subclasses with their respective superclass and with each other. Thus the large cochange leverage indicates inheritance relationships to complex superclasses across subsystem boundaries.

It could be argued that the chosen subsystem decomposition is inappropriate, and that uml.diagram should be considered as a single subsystem. From this perspective, the cochange leverage and its visualization hint at an improved decomposition. (The package uml.diagram is not considered as a single subsystem here because it is very large, with 231 source files.)

Table 10.2b shows the most leveraged subsystem pairs outside uml.diagram, which are discussed in the following.

{application, i18n} The class Main strongly cochanges with the class Translator. Although the class Main invokes methods of Translator, the structural relationship does not appear to be unusually strong, and thus none of the above-mentioned criteria for design flaws is fulfilled.

{kernel, uml.uml} The classes Project and ProjectMember strongly cochange with ProjectMemberModel. The class ProjectMember is a superclass of ProjectMemberModel. The classes Project and ProjectMemberModel are cyclicly related, they invoke methods of each other. Thus the large cochange leverage indicates similar responsibilities and strong structural relationships.

{language, uml.generator} The class GeneratorJava strongly cochanges with FileGenerator and the fairly large Generator2, which are its superclasses. Thus the large cochange leverage indicates strong inheritance relations across subsystem boundaries.

{kernel, persistence} The class Project strongly cochanges with several classes that are responsible for saving and loading projects. Thus the large cochange leverage indicates similar responsibilities which also induce an unusually large number of structural relationships (method invocations).

{kernel, uml.ui.ui} The classes Project and ProjectManager strongly cochange with ActionOpenProject, ActionSaveProject, ActionSaveProjectAs. The responsibilities are similar, but justify the assignment of the classes to different packages, and the structural relationships are not unusually strong.

{uml.notation, notation} The cochange strength is mostly contributed by the cochange of various classes in uml.notation with NotationProviderFactory2, which defines a numeric constant for each of these classes. Thus the large cochange leverage indicates duplicated information that must be kept consistent.

{uml.diagram.static_structure, kernel} The cochange strength is mostly contributed by the cochange of various classes in uml.diagram.static_structure with the class Project. The two packages have mutual structural relationships, but these are rather weak. Thus none of the criteria for design flaws appears to be satisfied.

{uml.cognitive, pattern} The cochange strength is mostly contributed by the coupling of classes that implement so-called critics. (Critics scan diagrams or models for potential design problems.) Thus the large cochange leverage indicates very similar responsibilities; the package pattern, whose only three classes are all critics, could be joined with the package uml.cognitive.critics.

{ui, kernel} The class ProjectBrowser strongly cochanges with the classes ProjectManager and Project. These classes have similar responsibilities and many structural relationships.

The class ArgoDiagram strongly cochanges with the class Project, which manages the diagrams and thus has strong structural relationships to ArgoDiagram.

The class TargetManager strongly cochanges with the class Project. These classes have mutual structural relationships (method invocations).

The class GenericArgoMenuBar strongly cochanges with the ProjectManager. For this pair of classes, none of the above criteria for design problems appears to be satisfied.

{language, uml.reveng} The class GeneratorJava strongly cochanges with the class Modeller. While GeneratorJava transforms UML models into Java source code, Modeller transforms Java source code into UML models. Thus the two classes have similar responsibilities, and include concepts that need to be kept consistent. Interestingly, the classes have no direct structural relationships.

Results for Eclipse jdt.ui The visualizations of the cochange graph are shown on page 173, and selected subsystem pairs with large cochange leverage are listed on page 172. Like for ArgoUML, the subsystems are packages, and the software elements are source code files which mostly coincide with top-level classes.

{internal.ui.jarpackager, ui.jarpackager} Packages in Eclipse plug-ins are classified into API packages and non-API packages; the earlier are intended for use by other plug-ins, and the latter contain internal implementation details. For many API packages, like ui.jarpackager, exists a corresponding non-API package, like internal.ui.jarpackager, with the same responsibilities.

Six further pairs of an API package and a corresponding non-API package are easily identified in the matrix in Figure 10.5b, because each pair is placed at neighboring positions (except ui.actions and internal.ui.actions, which are separated by one package). All of these pairs have a cochange leverage above 4, and are among the top 4% of the most leveraged subsystem pairs (see Table 10.3). In the following, each of these pairs is considered as a single subsystem.

Of course, the strong dependency between each API package and the corresponding non-API package is well-known to every developer of Eclipse plug-ins. The crucial point here is that the cochange leverage and its visualizations indeed reflect such strong dependencies, which are potential design flaws.

Results for Eclipse jdt.ui (API and non-API joined) The subsystem pairs with large cochange leverage and the visualizations for the adapted subsystem decomposition are shown on page 174 and 175.

{ui.packageview, ui.browsing} Both packages provide views on Java programs with similar functionality. The cochange strength is mostly contributed by the main classes of these views, i.e. by the cochange of PackageExplorerPart with JavaBrowsingPart, MembersView, PackagesView, ProjectsView, and TypesView, which are all subclasses of ViewPart. Furthermore, PackageExplorerContentProvider cochanges strongly with JavaBrowsingContentProvider; both subclass the fairly large StandardJavaElementContentProvider. Thus the large cochange leverage indicates parallel functionality with parallel implementations.

{corext.codemanipulation, corext.template} The class StubUtility cochanges strongly with CodeTemplateContextType and CodeTemplateContext. The class CodeTemplateContextType defines a large number of string constants, which are used more than 60 times in StubUtility, mostly in invocations of methods of CodeTemplateContext. Thus the large cochange leverage indicates extremely strong structural relationships.

{corext.codemanipulation, corext.util} A large part of the cochange strength is contributed by common changes with the large class corext.util.JavaModelUtil, which is also the target of many structural relationships. However, the common changes do not appear to indicate a serious design flaw.

{ui.packageview, ui.dnd} There are four inheritance relationships from classes in ui.packageview to classes in ui.dnd, and the class PackageExplorerPart in ui.packageview instantiates three classes from ui.dnd; overall, ui.dnd contains only six classes. Thus the large cochange leverage indicates strong structural relationships.

{ui.packageview, ui.workingsets} The package explorer view (implemented in ui.packageview) provides some working set functionality (implemented in ui.workingsets), and accordingly there are some structural relationships. However, the large cochange leverage does not appear to indicate a design flaw.

{ui.typehierarchy, ui.browsing} Both packages provide views on Java programs, and the cochange strength is mostly contributed by the main classes of these views, i.e. by the cochange of TypeHierarchyViewPart primarily with JavaBrowsingPart, but also with MembersView, PackagesView, ProjectsView, and TypesView. Furthermore, HierarchyLabelProvider cochanges strongly with PackagesViewLabelProvider; both are subclasses of the fairly large class (if inherited methods are included) AppearanceAwareLabelProvider. Thus the large cochange leverage indicates parallel functionality with parallel implementations.

{ui.packageview, ui.typehierarchy} Similar to {ui.packageview, ui.browsing} and {ui.typehierarchy, ui.browsing}.

{ui.typehierarchy, ui.util} Most of the cochange strength is contributed by common changes with the class OpenTypeHierarchyUtil in ui.util. Thus the large cochange leverage indicates that a class could be moved to a different package which better suites its responsibilities.

{ui.typehierarchy, ui.viewsupport} The package ui.typehierarchy provides a particular view on Java programs, while the package ui.viewsupport provides general classes for the support of such views. There are seven inheritance relationships from classes in ui.typehierarchy to classes in ui.viewsupport (four direct, two indirect, one interface implementation), and several other structural relationships. Thus the large cochange leverage indicates strong structural relationships.

{ui.callhierarchy, corext.callhierarchy} As the names indicate, the large cochange leverage indicates very similar responsibilities.

{ui.dialogs, corext.util} Most of the cochange strength is contributed by common changes of the class TypeInfoViewer with TypeInfoHistory, TypeInfoFilter, etc. Thus the large cochange leverage indicates that the implementation of a coherent concept has been dispersed over two packages.

{ui.browsing, ui.ui} The class JavaBrowsingPerspectiveFactory cochanges very strongly with the classes JavaPerspectiveFactory and JavaHierarchyPerspectiveFactory. These three classes not only implement the same interface IPerspectiveFactory, but consist largely of identical code.

The next subsystem pairs in Table 10.4 are mostly pairs of a client package and a supplier package, apparently without serious design flaws.

Discussion The results support both hypotheses, namely the existence of system parts with large cochange leverage, and the validity of the cochange leverage as indicator of design flaws. However, the precision of the cochange leverage is clearly not optimal; in particular, several pairs of a client subsystem and a supplier subsystem had a large cochange leverage although they were not flawed. On the other hand, fairly diverse and often nonobvious design flaws were detected, e.g. cyclic structural relationship between subsystems, inheritance from complex superclasses across

subsystem boundaries, inheritance from common complex superclasses in different subsystems, software elements with similar responsibilities in different subsystems, and other not purely textual similarities between different subsystems.

The precision of cochange leverage was not evaluated quantitatively, because the "flawedness" of subsystem pairs is difficult to quantify or classify. A simple classification of entire subsystem pairs into flaws and non-flaws appears to be too coarse-grained and would be fairly arbitrary in some cases. Previous empirical studies of cochange-based methods for detecting design flaws are also qualitative ([Kra03, Chapter 6], [GJK03], [IHL04], [DL06]); in contrast to the present study, they are limited to a small part of a software system, and do not apply defined criteria for identifying potential flaws and for assessing whether they are actually flaws.

The recall of the cochange leverage, i.e. the fraction of highly leverage system parts among the flawed system parts, was not evaluated. In general, cochange-based techniques can reliably detect design flaws only if they are involved in a sufficient number of logical changes; but flaws that are not involved in future logical changes are probably not worth refactoring anyway. The validity of the total cochange coupling as indicator of design quality, as defined in Subsection 9.1.2, only requires a high precision of the cochange leverage, not a high recall.

Threats to Validity The results of the study have been affected by the several parameters, whose impact is discussed in the following: the examined software systems, the chosen subsystem decompositions, and the method for predicting future cochange strengths (which together determine the cochange graphs), the examined type of system part, and the classification of the system parts as design flaws.

Only two *software systems* were examined, which are similar in many respects (programming language, open source, functionality). The results might not generalize to other types of systems.

The chosen *subsystem decompositions* were coarse grained. For more fine-grained decompositions, there are two alternatives: If the same minimum level for the cochange strength is retained, then a larger fraction of the subsystem pairs is filtered out; if the minimum level is reduced, then the precision of cochange leverage as indicator of design flaws will likely decrease, because not every single nonlocal logical change is caused by a design flaw, and thus a certain number of logical changes is required to infer flaws reliably. Concerning the first hypothesis, fine-grained decompositions contain system parts with even larger cochange leverage.

The used method for *predicting future cochange strengths* has been very simple (see Appendix A.2): Basically, past logical changes were projected into the future, and even the past logical changes were only derived heuristically from version repositories. More precise predictions may improve the precision of cochange leverage.

The examined *system parts* were subsystem pairs; the alternatives are similar to different subsystem granularities, as discussed in Subsection 10.3.1.

The *classification* of the detected pairs of subsystems or software elements as design flaws was performed by the author. The impact of subjectivity has been reduced, but clearly not eliminated, by defining and applying criteria for this classification.

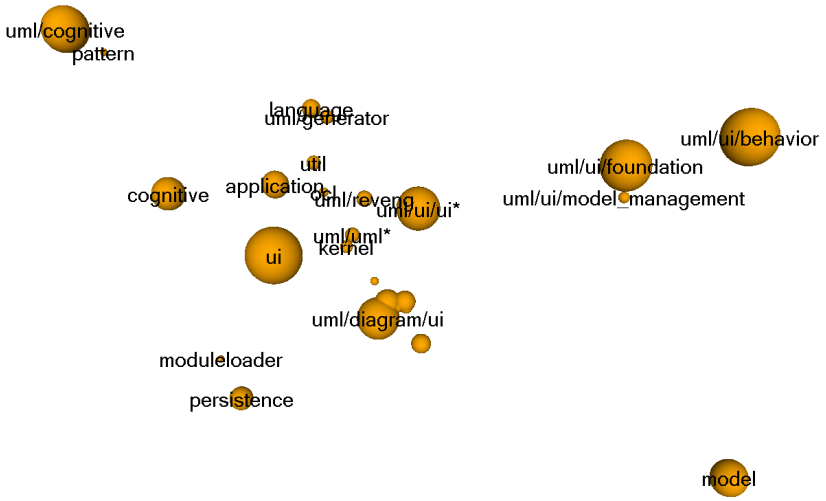
Subsystem Pair	Size of Pair		Caused Coupling		Leverage	
	This	Total	This	Total	This	Total
{kernel, uml.diagram.diagram}	36 (0.00%)	0.00%	15.5 (0.30%)	0.30%	63.5	63.5
{uml.diagram.collaboration, uml.diagram.deployment}	98 (0.01%)	0.02%	40.7 (0.80%)	1.10%	61.4	61.9
{application, i18n}	41 (0.01%)	0.02%	15.6 (0.31%)	1.41%	56.0	60.5
{uml.diagram.collaboration, uml.diagram.use_case}	70 (0.01%)	0.03%	24.5 (0.48%)	1.89%	51.6	58.0
{i18n, moduleloader}	3 (0.00%)	0.03%	1.0 (0.02%)	1.91%	49.2	57.9
{uml.diagram.deployment, uml.diagram.use_case}	140 (0.02%)	0.05%	42.1 (0.83%)	2.74%	44.4	53.0
{uml.diagram.collaboration, uml.diagram.activity}	70 (0.01%)	0.06%	14.7 (0.29%)	3.03%	30.9	49.6
{kernel, uml.uml}	108 (0.01%)	0.08%	21.4 (0.42%)	3.45%	29.2	45.8
{uml.diagram.static_structure, uml.diagram.use_case}	370 (0.05%)	0.12%	69.7 (1.37%)	4.81%	27.8	38.7
{uml.diagram.state, uml.diagram.activity}	250 (0.03%)	0.16%	42.0 (0.83%)	5.64%	24.8	35.7
{uml.diagram.collaboration, uml.diagram.state}	175 (0.02%)	0.18%	28.7 (0.56%)	6.20%	24.2	34.3
{uml.diagram.deployment, uml.diagram.activity}	140 (0.02%)	0.20%	22.8 (0.45%)	6.65%	24.1	33.3
{uml.diagram.use_case, uml.diagram.activity}	100 (0.01%)	0.21%	14.6 (0.29%)	6.94%	21.5	32.6
{uml.diagram.deployment, uml.diagram.state}	350 (0.05%)	0.26%	48.1 (0.94%)	7.88%	20.3	30.4
{uml.diagram.deployment, uml.diagram.static_str}	518 (0.07%)	0.33%	70.6 (1.39%)	9.27%	20.1	28.2
{uml.diagram.collaboration, uml.diagram.static_str}	259 (0.03%)	0.36%	34.6 (0.68%)	9.95%	19.7	27.4
{uml.diagram.state, uml.diagram.use_case}	250 (0.03%)	0.40%	31.7 (0.62%)	10.6%	18.7	26.7
{uml.diagram.collaboration, uml.diagram.diagram}	28 (0.00%)	0.40%	3.0 (0.06%)	10.6%	15.9	26.6
{uml.uml, uml.diagram.diagram}	48 (0.01%)	0.41%	5.1 (0.10%)	10.7%	15.7	26.4
{uml.diagram.state, uml.diagram.diagram}	100 (0.01%)	0.42%	10.2 (0.20%)	10.9%	15.1	26.1
{uml.diagram.use_case, uml.diagram.diagram}	40 (0.01%)	0.42%	3.5 (0.07%)	11.0%	13.0	25.9
{uml.diagram.static_structure, uml.diagram.diagram}	148 (0.02%)	0.44%	12.8 (0.25%)	11.3%	12.8	25.3
{ocl, uml.diagram.diagram}	20 (0.00%)	0.45%	1.6 (0.03%)	11.3%	11.8	25.2
{uml.diagram.collaboration, uml.diagram.sequence}	147 (0.02%)	0.47%	11.7 (0.23%)	11.5%	11.7	24.7
{uml.diagram.static_structure, uml.diagram.activity}	370 (0.05%)	0.52%	28.4 (0.56%)	12.1%	11.3	23.4
...	...	...	...	...	...	...

(a) Including subsystem pairs within uml.diagram

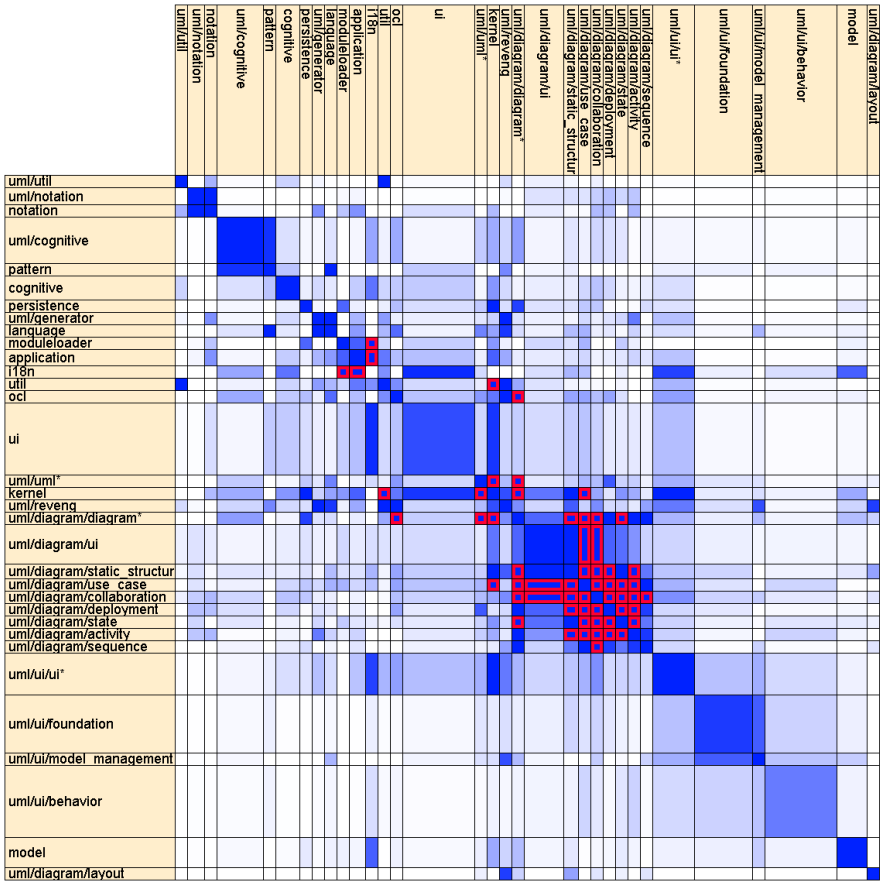
Subsystem Pair	Size of Pair		Caused Coupling		Leverage	
	This	Total	This	Total	This	Total
{kernel, uml.diagram.diagram}	36 (0.00%)	0.00%	15.5 (0.30%)	0.30%	63.5	63.5
{application, i18n}	41 (0.01%)	0.01%	15.6 (0.31%)	0.61%	56.0	59.5
{i18n, moduleloader}	3 (0.00%)	0.01%	1.0 (0.02%)	0.63%	49.2	59.1
{kernel, uml.uml}	108 (0.01%)	0.03%	21.4 (0.42%)	1.05%	29.2	41.9
{uml.uml, uml.diagram.diagram}	48 (0.01%)	0.03%	5.1 (0.10%)	1.15%	15.7	36.6
{ocl, uml.diagram.diagram}	20 (0.00%)	0.03%	1.6 (0.03%)	1.18%	11.8	34.7
{uml.diagram.use_case, kernel}	90 (0.01%)	0.05%	6.8 (0.13%)	1.32%	11.2	28.6
{kernel, util}	99 (0.01%)	0.06%	7.2 (0.14%)	1.46%	10.7	24.6
{language, uml.generator}	209 (0.03%)	0.09%	13.9 (0.27%)	1.73%	9.84	19.9
{kernel, persistence}	270 (0.04%)	0.12%	16.3 (0.32%)	2.05%	8.92	16.7
{kernel, uml.ui.ui}	936 (0.12%)	0.25%	56.1 (1.10%)	3.15%	8.85	12.7
{ocl, uml.reveng}	70 (0.01%)	0.26%	4.1 (0.08%)	3.23%	8.55	12.6
{language, pattern}	57 (0.01%)	0.26%	3.3 (0.06%)	3.30%	8.46	12.5
{uml.reveng, util}	154 (0.02%)	0.28%	8.3 (0.16%)	3.46%	7.91	12.1
{uml.notation, notation}	344 (0.05%)	0.33%	16.9 (0.33%)	3.79%	7.23	11.5
{uml.generator, uml.reveng}	154 (0.02%)	0.35%	7.1 (0.14%)	3.93%	6.83	11.2
{uml.util, util}	44 (0.01%)	0.36%	2.0 (0.04%)	3.97%	6.71	11.1
{uml.diagram.static_structure, kernel}	333 (0.04%)	0.40%	15.1 (0.30%)	4.26%	6.68	10.6
{ui, i18n}	178 (0.02%)	0.42%	7.7 (0.15%)	4.42%	6.39	10.4
{uml.cognitive, pattern}	345 (0.05%)	0.47%	14.6 (0.29%)	4.70%	6.23	9.99
{ui, kernel}	1602 (0.21%)	0.68%	66.1 (1.30%)	6.00%	6.09	8.77
{language, uml.reveng}	266 (0.04%)	0.72%	10.9 (0.21%)	6.21%	6.05	8.64
{uml.diagram.layout, uml.reveng}	84 (0.01%)	0.73%	3.3 (0.07%)	6.28%	5.86	8.60
{persistence, uml.diagram.diagram}	120 (0.02%)	0.75%	4.7 (0.09%)	6.37%	5.77	8.53
{i18n, uml.ui.ui}	104 (0.01%)	0.76%	4.0 (0.08%)	6.45%	5.73	8.48
...	...	...	...	...	...	...

(b) Excluding subsystem pairs within uml.diagram

Table 10.2: Subsystem pairs with large cochange leverage in ArgoUML



(a) Layout with small normalized atedge length. The subsystems notation, uml.notation (close together), uml.diagram.layout, and uml.util are far away from the other subsystems and not shown; the unnamed subsystems around uml.diagram.ui are the other subpackages of uml.diagram; subsystem i18n is overlaid by subsystem application.

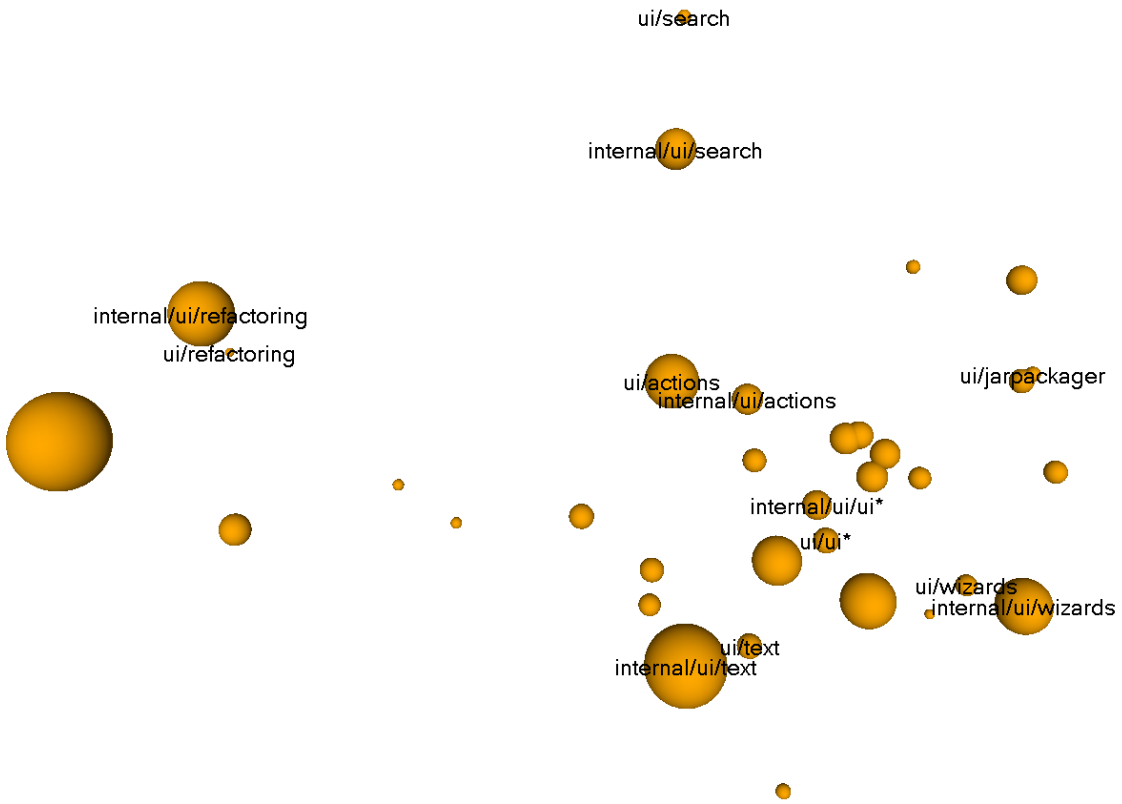


(b) Ordering with small normalized atedge length; the 29 subsystem pairs with the largest cochange leverage (0.76% of all subsystem pairs) are marked.

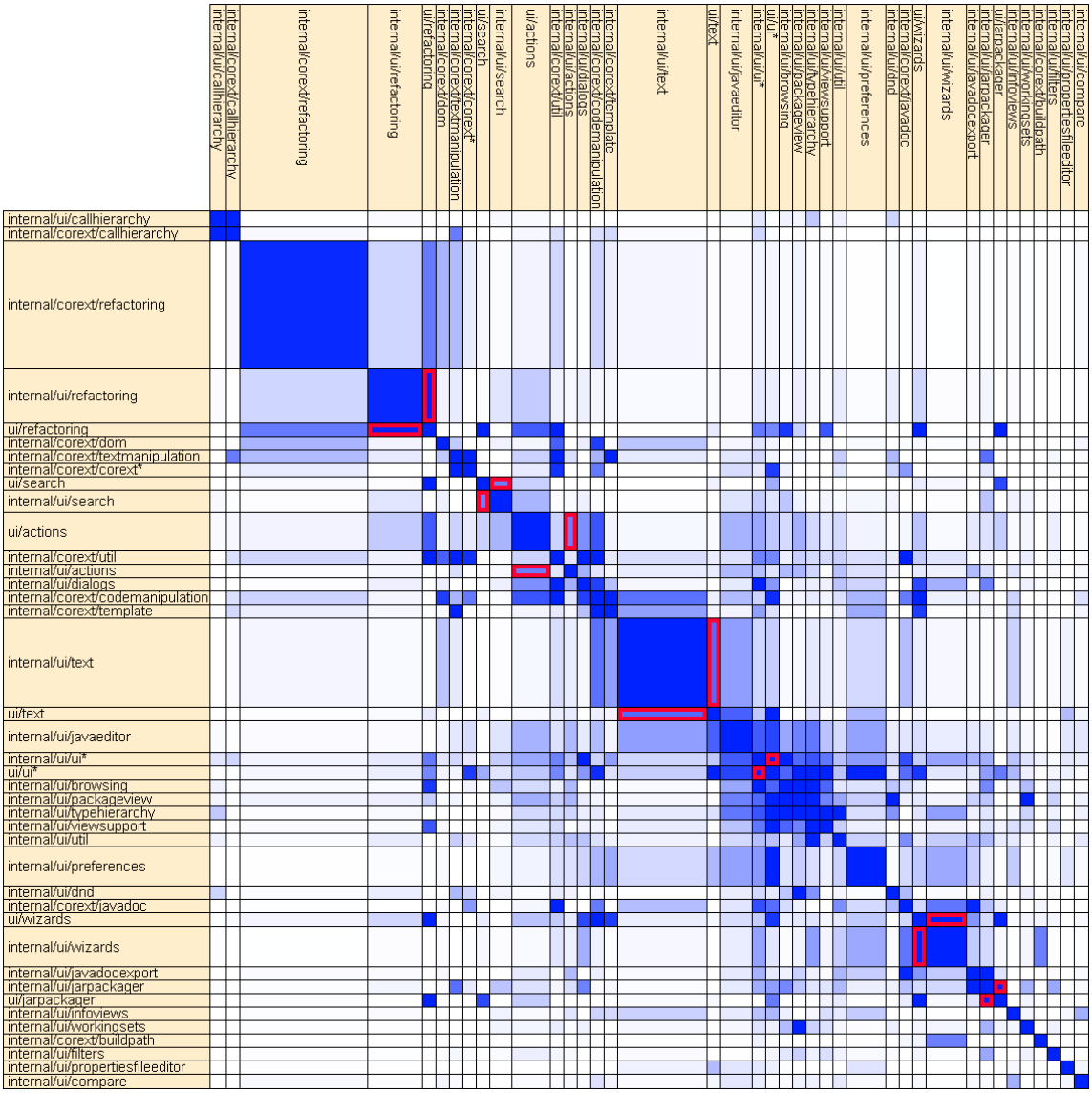
Figure 10.4: Cochange dependencies in ArgoUML

Subsystem Pair	Size of Pair		Caused Coupling		Leverage	
	This	Total	This	Total	This	Total
{internal.ui.jarpackager, ui.jarpackager}	126 (0.01%)	0.01%	24.6 (0.38%)	0.38%	33.0	33.0
...	...	...	...	...	...	...
{internal.ui.refactoring, ui.refactoring}	256 (0.02%)	0.21%	17.6 (0.27%)	3.24%	11.6	15.2
{internal.ui.wizards, ui.wizards}	1316 (0.12%)	0.33%	90.2 (1.39%)	4.63%	11.6	13.9
{internal.ui.ui, ui.ui}	520 (0.05%)	0.38%	34.6 (0.53%)	5.16%	11.2	13.6
...	...	...	...	...	...	...
{internal.ui.text, ui.text}	3933 (0.36%)	2.75%	128.0 (1.97%)	22.6%	5.50	8.22
...	...	...	...	...	...	...
{internal.ui.actions, ui.actions}	2610 (0.24%)	3.71%	73.6 (1.13%)	27.4%	4.76	7.39
...	...	...	...	...	...	...
{internal.ui.search, ui.search}	306 (0.03%)	3.91%	8.5 (0.13%)	28.4%	4.70	7.26
...	...	...	...	...	...	...

Table 10.3: Leveraged subsystem pairs in Eclipse jdt.ui (API and non-API separated)



(a) Layout with small normalized atedge length. Six subsystems are far away from the other subsystems and not shown; the unnamed subsystem near ui.jarpackager is internal.ui.jarpackager.

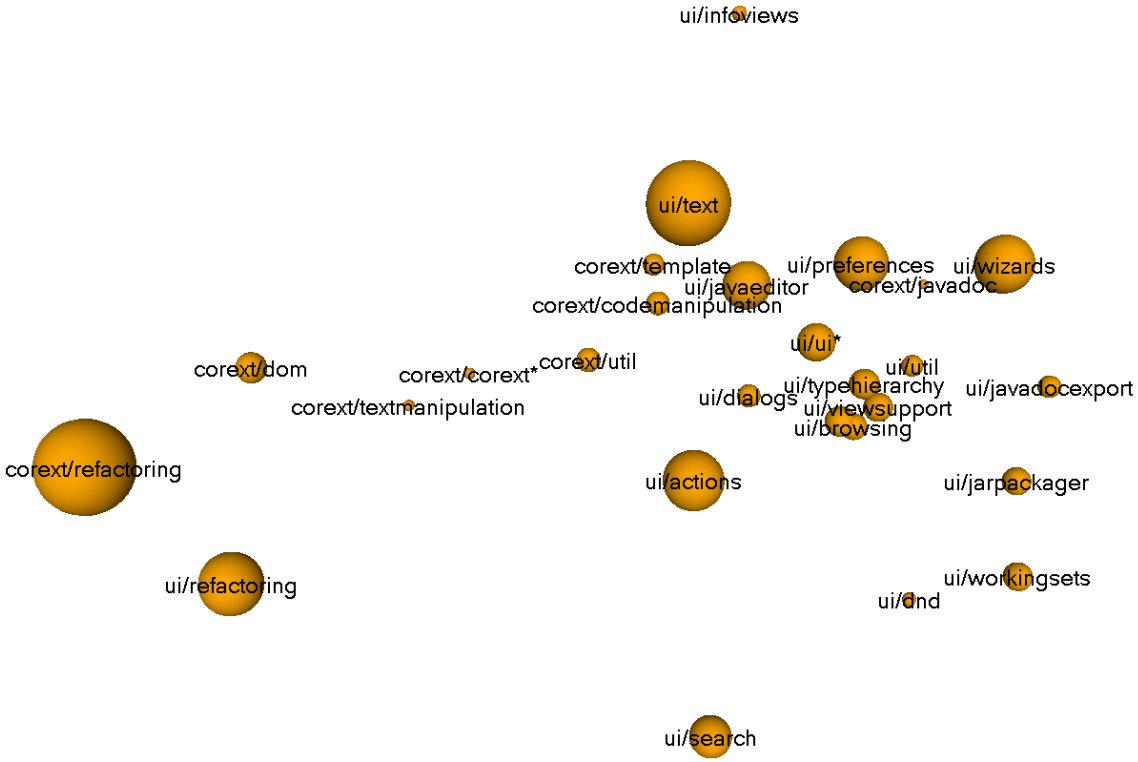


(b) Ordering with small normalized atedge length; the subsystem pairs of Table 10.3 are marked.

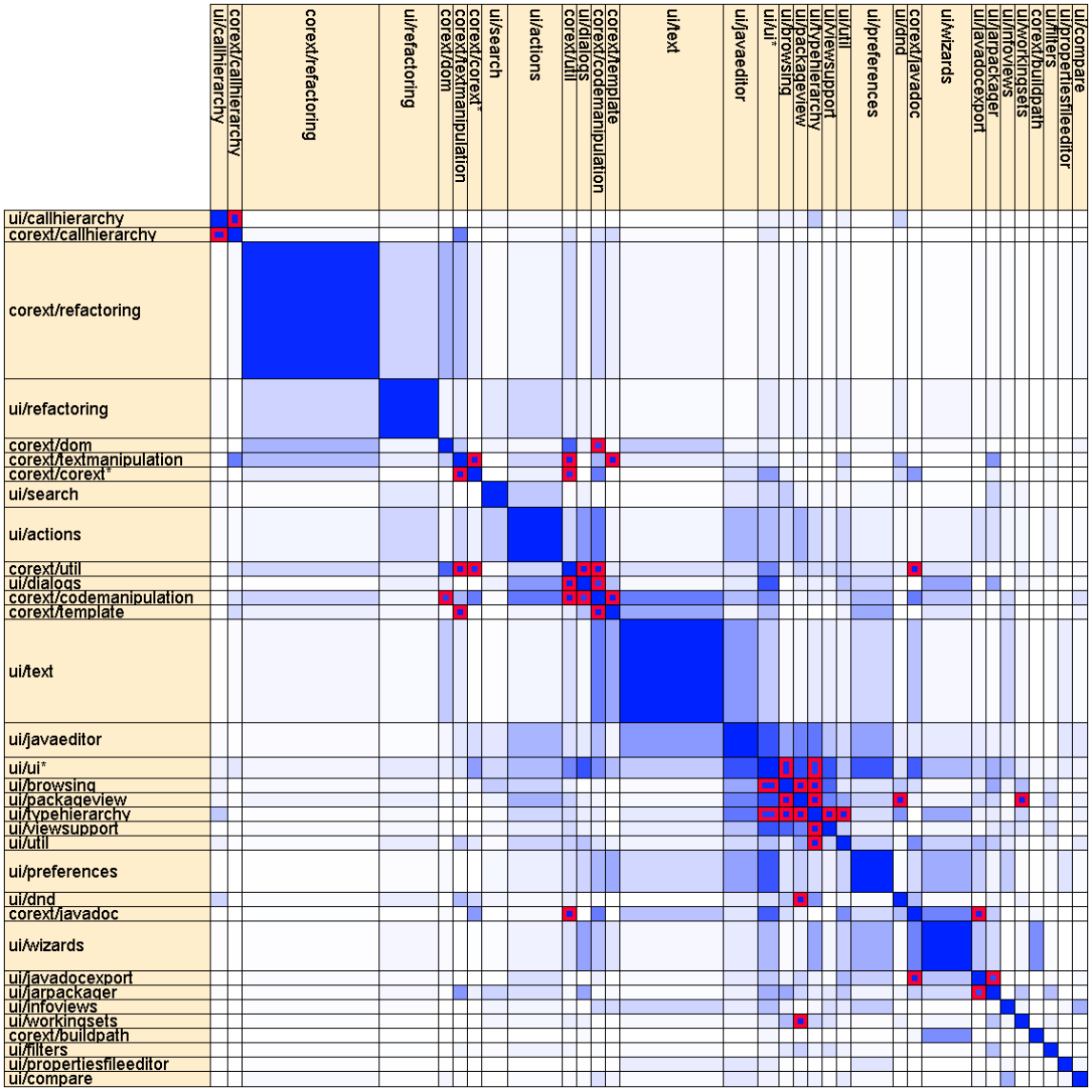
Figure 10.5: Cochange dependencies in Eclipse jdt.ui (API and non-API separated)

Subsystem Pair	Size of Pair		Caused Coupling		Leverage	
	This	Total	This	Total	This	Total
{ui.packageview, ui.browsing}	690 (0.06%)	0.06%	66.0 (1.08%)	1.08%	17.4	17.4
{corext.codemanipulation, corext.template}	270 (0.02%)	0.09%	25.4 (0.41%)	1.49%	17.0	17.3
{corext.textmanipulation, corext.util}	76 (0.01%)	0.09%	6.5 (0.11%)	1.60%	15.6	17.2
{corext.util, corext.javadoc}	57 (0.01%)	0.10%	4.8 (0.08%)	1.68%	15.3	17.1
{corext.codemanipulation, corext.util}	342 (0.03%)	0.13%	24.3 (0.40%)	2.08%	12.9	16.1
{ui.packageview, ui.dnd}	180 (0.02%)	0.15%	12.6 (0.21%)	2.28%	12.7	15.7
{ui.packageview, ui.workingsets}	780 (0.07%)	0.22%	48.7 (0.80%)	3.08%	11.3	14.3
{corext.textmanipulation, corext.corext}	16 (0.00%)	0.22%	1.0 (0.02%)	3.10%	11.1	14.3
{ui.typehierarchy, ui.browsing}	667 (0.06%)	0.28%	40.6 (0.66%)	3.76%	11.0	13.6
{ui.packageview, ui.typehierarchy}	870 (0.08%)	0.36%	52.2 (0.85%)	4.61%	10.9	13.0
{ui.typehierarchy, ui.util}	435 (0.04%)	0.39%	25.6 (0.42%)	5.03%	10.7	12.7
{ui.typehierarchy, ui.viewsupport}	783 (0.07%)	0.47%	43.5 (0.71%)	5.74%	10.1	12.3
{corext.textmanipulation, corext.template}	60 (0.01%)	0.47%	3.3 (0.05%)	5.80%	10.0	12.3
{ui.callhierarchy, corext.callhierarchy}	518 (0.05%)	0.52%	28.4 (0.46%)	6.26%	9.96	12.1
{ui.dialogs, corext.util}	323 (0.03%)	0.55%	17.5 (0.29%)	6.55%	9.86	12.0
{ui.javadocexport, corext.javadoc}	48 (0.00%)	0.55%	2.3 (0.04%)	6.59%	8.57	12.0
{ui.browsing, ui.ui}	1058 (0.10%)	0.65%	48.8 (0.80%)	7.38%	8.38	11.4
{corext.util, corext.corext}	76 (0.01%)	0.65%	3.4 (0.06%)	7.44%	8.17	11.4
{ui.typehierarchy, ui.ui}	1334 (0.12%)	0.77%	57.0 (0.93%)	8.37%	7.75	10.8
{corext.codemanipulation, corext.dom}	558 (0.05%)	0.82%	23.5 (0.38%)	8.76%	7.65	10.6
{ui.dialogs, corext.codemanipulation}	306 (0.03%)	0.85%	12.8 (0.21%)	8.97%	7.56	10.5
{ui.javadocexport, ui.jarpackager}	400 (0.04%)	0.89%	15.8 (0.26%)	9.22%	7.17	10.4
{ui.packageview, ui.ui}	1380 (0.12%)	1.01%	54.4 (0.89%)	10.1%	7.15	10.0
{ui.dialogs, ui.ui}	782 (0.07%)	1.08%	29.7 (0.49%)	10.6%	6.90	9.80
{ui.javaeditor, ui.ui}	3404 (0.31%)	1.39%	129.3 (2.11%)	12.7%	6.89	9.16
...	...	...	...	...	...	...

Table 10.4: Leveraged subsystem pairs in Eclipse jdt.ui (API and non-API joined)



(a) Layout with small normalized atedge length. The subsystems corext.callhierarchy, ui.callhierarchy (close together), corext.buildpath, ui.compare, ui.filters, ui.propertiesfileeditor are far away from the other subsystems and not shown; the unnamed subsystem near ui.browsing is ui.packageview.



(b) Ordering with small normalized atedge length; the 22 subsystem pairs with the largest cochange leverage of Table 10.4 (0.89% of subsystem pairs) are marked.

Figure 10.6: Cochange dependencies in Eclipse jdt.ui (API and non-API joined)

10.3.4 Related Work

Coupling Measures The main result of this section is the cochange leverage (or equivalently, the cochange density) as indicator of cochange-related design flaws. In the following, the cochange leverage is compared with existing measures for the cochange coupling between software elements or subsystems.

The comparison concerns only the suitability for detecting design flaws, and not the suitability for other purposes, like predicting modifications. The existing measures were originally defined for graph models where the vertices are software elements (not subsystems), and the edge weights are cochange counts (not cochange strengths)[6]; here all measures are uniformly applied to cochange graphs (as defined in Section 10.1), to make the results comparable.

Indicators of problematic dependencies should not be biased towards small or large subsystems, because subsystem size is not related to quality, but only to the chosen level of detail. (Even if size is related to quality, size and coupling should be measured separately.) Each coupling measure is checked for such biases theoretically and empirically. As theoretical check, it is calculated whether the measurement values depend on the subsystem sizes for cochange graphs with uniform density, i.e. where all pairs of software elements have the same cochange strength. The empirical check is facilitated by the matrix visualizations in Figures 10.7 (for ArgoUML) and 10.8 (for Eclipse jdt.ui) on pages 178 and 179. The rows and columns represent subsystems ordered by increasing size, and the color density of each matrix element represents a measurement value. Biases towards certain subsystem sizes are reflected by an uneven distribution of the densely colored matrix elements.

The *cochange strength* has been used by many researchers [EW93, MW01, BAY03, GJK03, ZDZ03, ZWDZ05, BW03, Ger04, HH04a, YMNCC04, RFG05, FG06a, DL06], and has been named pair change coupling [BAY03], support [ZDZ03, YMNCC04, ZWDZ05], or intensity of coupling [RFG05]. In a cochange graph with uniform density, the cochange strength of each subsystem pair is proportional to the sizes of both subsystems. This bias is confirmed by Figures 10.7a and 10.8a, where the color density (cochange strength) strongly increases from top left (pairs of small subsystems) to bottom right (pairs of large subsystems). Note that the diagonal elements should be ignored because they do not represent cochange causes. A variation of the cochange strength by Stringfellow et al. [SAP+06] incorporates the number of modified lines of code, but does not reduce the bias.

The *cochange density* was introduced in Subsection 10.3.1, and is equivalent (up to a constant factor) to the cochange leverage. The cochange density between two subsystems $\{s_1, s_2\}$ is defined as the average cochange strength between their software elements, formally as $\frac{\mathrm{f}(\{s_1,s_2\})}{\mathrm{w}(s_1)\mathrm{w}(s_2)}$. For a cochange graph with uniform density, the cochange density is obviously fixed, and thus independent of the subsystem sizes. This absence of bias is confirmed by Figures 10.7b and 10.8b, where the matrix ele-

[6]The choice of cochange strength (not cochange count) does not affect the conclusions, because both weighting schemes are similar (see Subsection 10.1.3). The choice of subsystems (not software elements) emphasizes size-related biases, because the subsystem sizes are less uniform.

ments with the largest color density are distributed fairly uniformly over the matrix. A variation of the cochange density with similar properties is the average cochange strength caused by each pair of individual changes, formally $\frac{\mathrm{f}(\{s_1,s_2\})}{\frac{1}{2}\deg(s_1)\frac{1}{2}\deg(s_2)}$. It can be seen as a special case of the cochange density where the size $\mathrm{w}(s)$ of each subsystem s is its number of individual changes $\frac{1}{2}\deg(s)$.

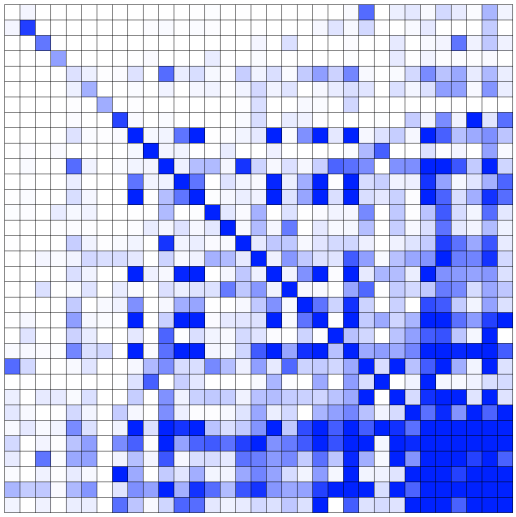
The *cochange probability* was introduced by Ball et al. [BKPS97] and also used by Eick et al. [EGK+02]. Ball et al. do not justify the name by specifying for which random event from which probability space their measure provides a probability. The cochange probability between two subsystems $\{s_1, s_2\}$ is defined as $\frac{\mathrm{f}(\{s_1,s_2\})}{\sqrt{\frac{1}{2}\deg(s_1)\frac{1}{2}\deg(s_2)}}$. For a cochange graph with uniform density, the cochange probability is proportional to $\sqrt{\frac{1}{2}\deg(s_1)\frac{1}{2}\deg(s_2)}$. Thus the cochange probability of two subsystems depends on their number of individual changes, which is usually correlated with their size. Figures 10.7c and 10.8c confirm that the bias is weaker than for the cochange strength in subfigures (a), but stronger than for the cochange density in subfigures (b).

The *cochange confidence* has been used by Zimmermann et al. [ZDZ03, ZWDZ05], by Hassan and Holt [HH04a], and as modification coupling by German [Ger04]. The cochange confidence of an ordered pair of subsystems (s_1, s_2) is defined as their average cochange strength per individual change of s_1, formally as $\frac{\mathrm{f}(\{s_1,s_2\})}{\frac{1}{2}\deg(s_1)}$. For a cochange graph with uniform density, the cochange confidence is proportional to $\frac{1}{2}\deg(s_2)$, i.e. to the number of individual changes of s_2, which is usually correlated with the size of s_2. This bias is reflected in Figures 10.7d and 10.8d, where the color density increases from left to right.

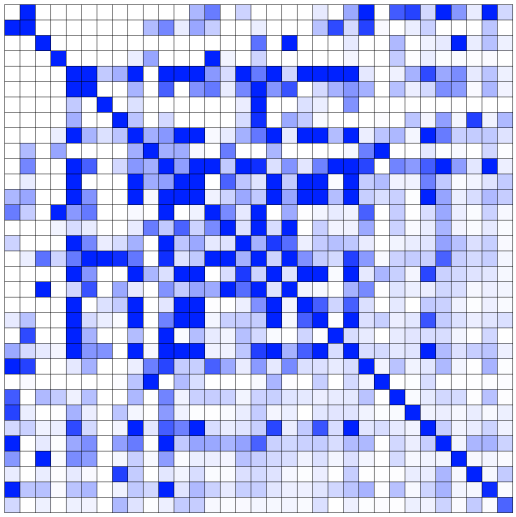
To summarize, the cochange strength, the cochange probability, and the cochange confidence between subsystems tend to increase with the size of the subsystems, and thus indicate not only design problems but also size; only the cochange density is not biased towards particular subsystem sizes.

Orderings In the matrix visualizations of cochange models by Zimmermann et al. [ZDZ03] and Burch et al. [BDW05], software elements (e.g. files) are ordered according to a given subsystem hierarchy (e.g. a directory tree), such that all software elements in the same subsystem have contiguous positions. Such hierarchical orderings support the identification of *pairs* of strongly cochanging software elements and subsystems at any level of the hierarchy, but generally not the identification of *groups* of strongly cochanging software elements or subsystems. These groups form clear visual patterns only if their members are placed at successive positions (confer Figures 10.3b and 10.4b), which is not ensured by hierarchical orderings.

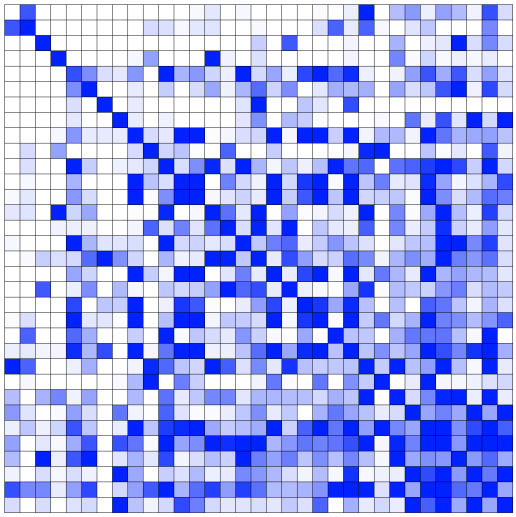
Layouts Several researchers have computed layouts of cochange models, and have interpreted closely placed vertices in these layouts as groups of strongly cochanging software elements, without providing evidence that the layouts actually permit such interpretations. Eick and Wills [EW93] and Ball et al. [BKPS97] generate their layouts with a custom energy model, but do not analyze the properties of this energy model. Burch et al. [BDW05] mention that their layouts are force-directed, with-



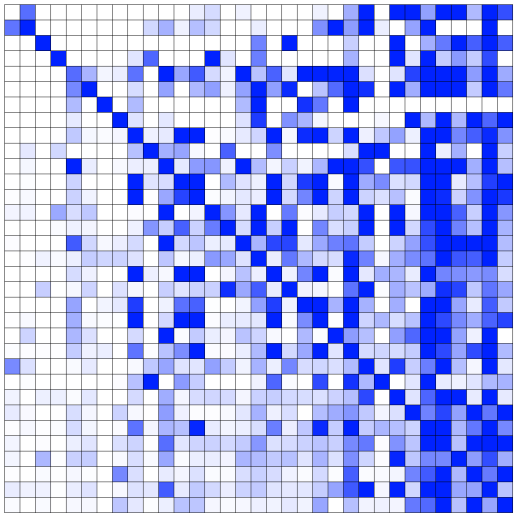
(a) Cochange strength



(b) Cochange density

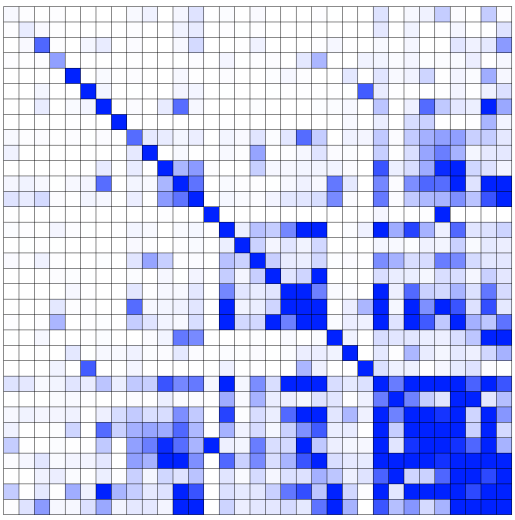


(c) Cochange probability

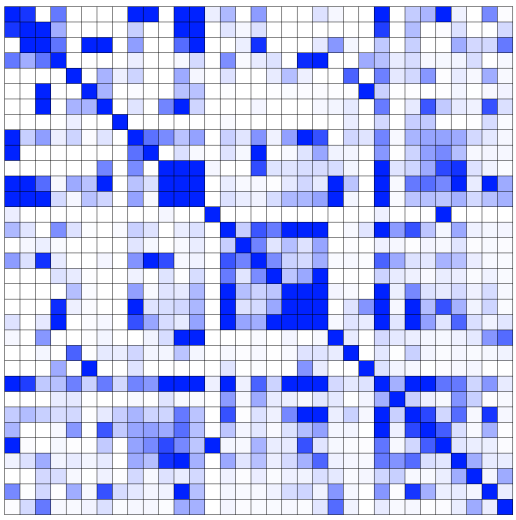


(d) Cochange confidence

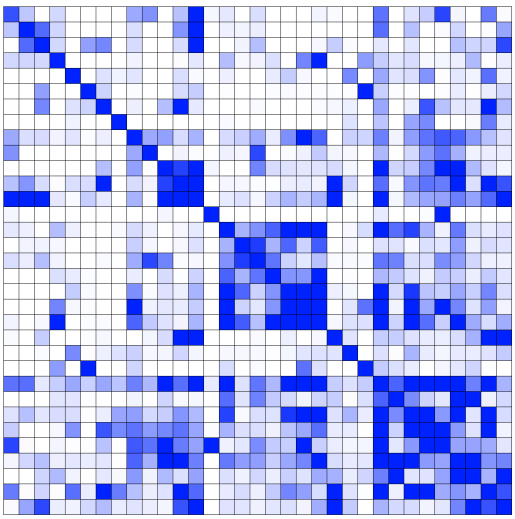
Figure 10.7: Coupling between the subsystems of ArgoUML, for four different coupling measures. The rows and columns represent subsystems, ordered by ascending size (from top to bottom and from left to right). The color density of each matrix element represents a measurement value. An uneven distribution of the densely colored matrix elements indicates a bias of the measure. (The row height and column width are constant, i.e. independent of subsystem size, for readability.)



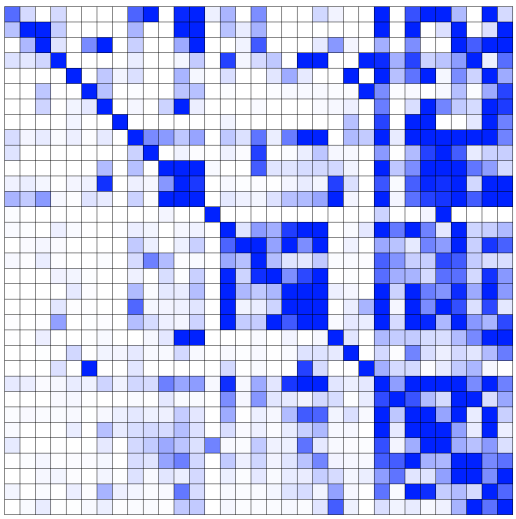
(a) Cochange strength



(b) Cochange density



(c) Cochange probability



(d) Cochange confidence

Figure 10.8: Coupling between the subsystems of Eclipse jdt.ui, for four different coupling measures. See the facing page for explanations.

out specifying the force model. Fischer and Gall [FG04] use a formula with several parameters to convert common changes into desired distances, and then apply multidimensional scaling to compute layouts from these distances; they do not analyze how the distances in the layouts can be interpreted in terms of the cochange model. The application of the LinLog energy model for layouting cochange models was first reported by the present author in a joint work with Beyer [BN05], and more recently by Fischer and Gall [FG06a] and Beyer and Hassan [BH06].

10.4 Summary

- While the (widely used) cochange count of two software elements is simply the number of their common logical changes, the (new) cochange strength also factors in the size of logical changes, and thereby avoids an overweighting of large changes.
- The $\overset{\text{total}}{\underset{\text{scaled}}{}}$ cochange coupling in a software system is the $\overset{\text{total}}{\underset{\text{average}}{}}$ cochange strength of software elements in different subsystems. The normalized cochange coupling is the average cochange strength of software elements in different subsystems, divided by the average cochange strength of all pairs of software elements.
- The $\overset{\text{total}}{\underset{\text{normalized}}{\text{scaled}}}$ cochange coupling equals the $\overset{\text{total}}{\underset{\text{normalized}}{\text{scaled}}}$ atedge cut of the cochange graph.
- The total and scaled cochange coupling are reflected in matrix visualizations.
- The scaled cochange coupling is not biased towards coarse or fine subsystem decompositions; the normalized cochange coupling is additionally not biased towards small or large sets of logical changes.
- The total cochange coupling is proportional to the additional development costs that occur because logical changes are nonlocal, under simplifying assumptions.
- Minimizing the total cochange coupling, as formalization of the design principle of locality of change (also called modularity), subsumes several existing design rules and design (anti)patterns.
- The cochange leverage of a subsystem pair is proportional to its inter-density.
- The cochange leverage is directly reflected in matrix visualizations. Densely cochanging subsystems are grouped by orderings and layouts with small normalized atedge length.
- The examined software systems contain subsystem pairs with large cochange leverage, i.e. a large part of the cochange coupling is caused by relatively few small subsystem pairs.
- In the examined systems, the precision of cochange leverage as indicator of design flaws is fairly high. (The recall was not evaluated.) The indicated flaws are nontrivial, e.g. delocalized responsibilities or strong structural relationships.
- Existing measures of pairwise cochange coupling, like support and confidence, indicate not (only) design flaws, but (also) subsystem sizes; cochange leverage is not biased towards particular subsystem sizes.

Chapter 11

D-Ref Coupling and D-Ref Leverage

Sometimes, *small* coupling is harmful. In version 1.4.2 of the Java Development Kit[1], only 1 of 46 classes in the package java.text references only 2 of 354 classes in the package java.awt – which suggests that java.text does not really need to reference java.awt at all. Indeed, software designers know that text formatting packages should generally be independent of user interface toolkits. But can such flawed references be detected automatically, without knowledge of the responsibilities of the packages?

This chapter introduces a measure of coupling (as indicator of design quality) and a measure of leverage (as indicator for design flaws) for d-ref dependencies, a particular type of dependency caused by references between subsystems. The first section introduces a model of references. The second section defines a measure of d-ref coupling, and argues that it is related to the development costs caused by inter-subsystem references. The third section derives a measure of d-ref leverage, and examines empirically whether references with large d-ref leverage are design flaws.

This chapter does *not* discuss how references between different subsystems can be removed without changing the external behavior of a software system. This is explained comprehensively in several existing books, e.g. [Lak96, Chapter 5], [Mar03, Chapter 20], and [RL06, Section 4.3].

11.1 References

This section defines a model of references between software elements and subsystems.

11.1.1 Definition

A software element or subsystem s_1 *references* a software element or subsystem s_2 if s_1 uses an identifier that is declared in s_2. Common types of references in object-oriented programs are (static) invocations of methods, accesses to attributes, inheritance of classes, and references to types e.g. in type casts and variable declarations.

[1] The software system JDK 1.4.2 and its model are described in Appendix A.3.

In the Java program in Figure 11.1a, for example, the class e_2 in the package s_1 references the classes e_3 and e_4 in the package s_2, and thus the package s_1 references the package s_2.

A mapping of software systems with a subsystem decomposition to graphs with a clustering is defined in the upper part of Table 11.1. The resulting cluster graph is called the *reference graph* of a software system. The actual construction of the reference graph requires specific definitions of the terms software element, subsystem, size of a software element, and size of an element reference; example definitions for source code in the programming language Java can be found in Appendix A.3. By the definition of cluster graph, the size of a subsystem is the sum of the sizes of its software elements, and the size of a subsystem reference is the sum of the sizes of the corresponding element references.

Reference graphs are *directed*, to capture the direction of the references; the graphs used in the previous chapters and defined in Section 2.2 are *undirected*. In a *directed graph* $G = ((V, \mathrm{w}), (E, \mathrm{f}))$, the edges are *ordered* pairs of vertices, i.e. $E \subseteq V^2$. Concepts that were defined for undirected graphs can be adapted to directed graphs by replacing unordered vertex pairs with the corresponding ordered vertex pairs.

Software System	Graph
software element	base vertex
size of software element	weight of base vertex
element reference	base edge
size of element reference	weight of base edge
subsystem	cluster
subsystem size	weight of cluster
subsystem reference	cluster edge
size of subsystem reference	weight of cluster edge
d-ref coupling of two subsystems	product of cluster weights
total d-ref coupling	total atpair cut of the cluster graph
scaled d-ref coupling	scaled atpair cut of the cluster graph
d-ref cause	cluster edge
size of d-ref cause	weight of cluster edge
leverage of d-ref cause	sparsity of cluster edge

Table 11.1: Graph model of references (upper part), d-ref coupling (central part), and d-ref leverage (lower part)

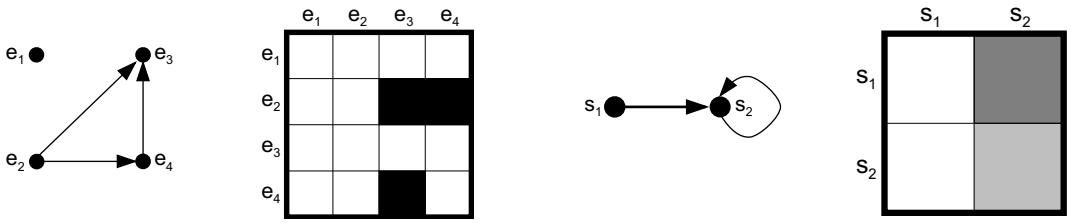
11.1.2 Visualization

Like any graph, reference graphs can be visualized as box-line diagram and as matrix (see Section 2.4). This chapter contains mostly matrix visualizations, because they reflect some relevant properties (defined in the next sections) more clearly. In matrix visualizations, each subsystem is represented by a row and a column, the subsystem size by the row height and the column width, each reference by the matrix element where the row of its initial subsystem and the column of its terminal subsystem intersect, and the size of the reference by the color weight in the matrix element.

For example, Figure 11.1e displays a matrix visualization of the reference graph for the Java program in Figure 11.1a, where all software elements are classes and have size 1, and all references between software elements have size 1. It shows that subsystem s_1 references subsystem s_2 but not vice versa, and that the references from s_1 to s_2 is larger than the reference from s_2 to s_2.

Package s_1: `class e1 { }`
`class e2 { e3 a = new e4(); }`
Package s_2: `class e3 { }`
`class e4 extends e3 { }`

(a) Four software elements in two subsystems



(b) Base graph (c) Base graph (d) Cluster graph (reference graph) (e) Cluster graph (reference graph)

Figure 11.1: Reference models for four software elements

11.1.3 Related Work

Models A large number of existing techniques and tools for the comprehension, evaluation, and evolution of software systems are based on references between software elements, and thus models of references are ubiquitous. The definition of the reference graph is not a significant contribution of this work, but only the basis for the next sections.

Visualizations Many previous works use matrices to visualize dependencies between design elements in general (e.g. Design Structure Matrices [Ste81, BC00, SGCH01]) and references between software subsystems in particular [BP01, vH03, SJSJ05, MRB06, LL07]. The matrices in this chapter are unique in their proportionality of certain software properties to visual properties, in particular of subsystem size to column size and row size, of reference size to color weight, and of reference density to color density. This proportionality enables the visual assessment of d-ref coupling and d-ref leverage discussed in the next sections.

11.2 D-Ref Coupling

This section introduces a measure of d-ref coupling (with two variants), and shows how its values are represented in matrix visualizations of the reference graph. The third subsection motivates the measure and its relation to design quality, by showing that it quantifies the additional comprehension and modification costs caused by references between subsystems, under simplifying assumptions.

11.2.1 Definition

A subsystem s_1 *d-ref depends* on another subsystem s_2 if s_1 references s_2. The *d-ref coupling* from a subsystem s_1 to a subsystem s_2 is the product of the sizes of s_1 and s_2 if s_1 d-ref depends on s_2, and is 0 otherwise. The *total d-ref coupling* in a software system is the sum of the d-ref couplings of all ordered pairs of different subsystems, or equivalently the total atpair cut of the reference graph[2]. In other words, the total d-ref coupling is the weighted number of references between different subsystems, where each reference is weighted with the size of the referencing and the referenced subsystem. The *scaled d-ref coupling* is the ratio of the total d-ref coupling to the maximum possible total d-ref coupling (i.e. to the total d-ref coupling if all subsystem pairs were d-ref dependent), or equivalently the scaled atpair cut of the reference graph. Clearly, the scaled d-ref coupling is 0 if no subsystem references any other subsystem, and 1 if every subsystem references every other subsystem (assuming there is more than one nonempty subsystem).

In Figure 11.1, the d-ref coupling from subsystem s_1 to subsystem s_2 is $2 \cdot 2 = 4$, the d-ref coupling from s_2 to s_1 is 0, thus the total d-ref coupling is $4 + 0 = 4$, and the scaled d-ref coupling is $\frac{4+0}{4+4} = \frac{1}{2}$.

It may be conjectured that most large software systems have a scaled d-ref coupling significantly smaller than 1, because they would be incomprehensible otherwise. This conjecture will not be verified here, but at least the five real-world software systems shown on pages 191 to 197 do not disprove it: Their scaled d-ref coupling is 0.525, 0.445, 0.447, 0.663, and 0.476, respectively.

As discussed in Chapter 7, the scaled d-ref coupling should not be compared for different systems or for different subsystem decompositions, because it is biased towards certain subsystem decompositions. However, the total d-ref coupling and the scaled d-ref coupling can be compared for different versions of the same system with the same subsystem decomposition; this will be exploited for detecting design flaws in Section 11.3.

11.2.2 Visualization

The total and scaled d-ref coupling in a software system are directly reflected in the matrix visualization of its reference graph, as detailed in Chapter 7 for the corresponding clustering quality measures. The d-ref coupling between two subsystems is the non-white area of the corresponding matrix element (which is 0 if the matrix element is white, and the area of the matrix element if it is non-white). The total d-ref coupling equals the total non-white area outside the diagonal elements, and the scaled d-ref coupling equals the fraction of non-white area outside the diagonal elements. In Figure 11.1e, the scaled d-ref coupling is $\frac{1}{2}$, and accordingly half of the off-diagonal area is non-white.

[2]The total atpair cut and the scaled atpair cut are clustering quality measures that were defined in Chapter 7.

11.2.3 Relation to Development Costs

This subsection shows, for a simple model of comprehension and modification, that the total d-ref coupling in a software system equals the d-ref costs for its comprehension and modification. *D-ref costs* are costs that are caused by the references between different subsystems; they do not include costs that are independent of references, and thus could not be avoided by removing references. It should be stressed that the used model of comprehension and modification is so simple that precise equality of d-ref coupling and d-ref costs cannot be expected in practice; the goal is merely to motivate the definition of the total d-ref coupling, and to provide evidence for a relation between the total d-ref coupling and design quality. The argumentation is presented for comprehension costs, but similarly applies to the costs of externally visible modifications, i.e. of modifications that affect the interface of a subsystem.

A *comprehension process* serves to comprehend a single subsystem called its *initial subsystem*, but may involve further subsystems as specified below. The following assumptions about comprehension processes are made:

1. A comprehension process with the initial subsystem s involves the comprehension of s itself and of all subsystems that s references. On the one hand, these subsystems are necessary, because the thorough comprehension of s requires the comprehension of all identifiers it uses, including those declared in other subsystems. On the other hand, these subsystems are sufficient if all identifiers are adequately documented in their declaring subsystem. An alternative assumption is explored in the next chapter.

2. The net costs of comprehending a subsystem s, excluding the costs of comprehending other subsystems referenced by s, are the size of s. In principle, this assumption can be easily satisfied by setting the size of each subsystem to the net costs of its comprehension. In practice, it may be difficult to determine these net costs, but the experiments in the next sections show that it may be sufficient to set the size simply to the number of contained software elements.

3. The frequency of comprehending a subsystem s, i.e. the number of comprehension processes whose initial subsystem is s, is its size. The discussion is similar to the previous point.

4. Each comprehension process is independent of the other comprehension processes, i.e. knowledge gained in a comprehension process is not reused in other comprehension processes. This could be correct if each comprehension process is performed by a different developer; otherwise it leads to an overestimation of the total costs.

Now the d-ref costs of comprehension can be easily calculated. Let $((V, \mathrm{w}), (E, \mathrm{f}))$ be the reference graph. According to assumptions 1 and 2, the costs of a single comprehension process with the initial subsystem s are the size of s plus the size of its referenced subsystems, i.e. $\sum_{(s,t)\in E\cup I} \mathrm{w}(t)$ (where I is the identity relation). Of these costs, the costs of comprehending s itself are not caused by references between

subsystems and thus do not belong to the d-ref costs. Thus the d-ref costs of a comprehension process with the initial subsystem s are $\sum_{(s,t)\in E\setminus I} \mathrm{w}(t)$. According to assumption 3, the subsystem s is the initial subsystem of $\mathrm{w}(s)$ comprehension processes, which are independent according to assumption 4, and thus have the total d-ref costs $\mathrm{w}(s) \sum_{(s,t)\in E\setminus I} \mathrm{w}(t)$. The total d-ref costs of the comprehension processes for all subsystems are $\sum_{s\in V} \left(\mathrm{w}(s) \sum_{(s,t)\in E\setminus I} \mathrm{w}(t)\right) = \sum_{(s,t)\in E\setminus I} \mathrm{w}(s)\mathrm{w}(t)$, which is the total atpair cut of the reference graph.

11.2.4 Related Work

Most coupling measures in the literature quantify the coupling between a single software element or subsystem and the remaining system, and not the total coupling of all subsystems in a software system. Such measures are potentially applicable for locating design flaws, and are therefore discussed in the next section.

The *coupling factor* in the MOOD metrics set [eAC94] for object-oriented systems is equivalent to the special case of the scaled atpair cut where the subsystems are classes and have weight 1. The cumulative component dependency of Lakos [Lak96, Section 4.12] and the indicator for regression testing costs of Leitch and Stroulia [LS03] are similar to the total d-ref coupling, but were proposed for the transitive closure of references, and are therefore discussed in the next chapter.

Many authors discuss the impact of references on comprehensibility, changeability, reusability, and testability (e.g. [Mar03, Chapters 11, 20], and [RL06, Chapter 3]), and several models of change propagation are based on references [KGH+95, BWL99, CKKL02, TCS05, Rob05]. However, none of these works derives a quantitative indicator for the total development costs caused by references.

11.3 D-Ref Leverage

This section derives a measure of leverage from the measure of d-ref coupling defined in the previous section, and examines its validity as indicator of design flaws.

11.3.1 Definition

In this section, references between different subsystems (shortly *inter-references*) are exemplarily considered as causes of d-ref coupling, and thus a measure of d-ref leverage is derived and evaluated for inter-references. A possible alternative would be to consider software elements or subsystems as causes of d-ref coupling, e.g. of the d-ref coupling that is induced by their outgoing references.

A reference from a subsystem s_1 to another subsystem s_2 increases the total d-ref coupling by the product of the sizes of s_1 and s_2. As defined in Section 9.1, a dependency cause has a leverage of $\frac{k_1}{k_2}$ if it causes $k_1\%$ of the total coupling, and its size is $k_2\%$ of the total size of all dependency causes. Thus, for a reference graph

$((V, \mathrm{w}), (E, \mathrm{f}))$, the *d-ref leverage* of an inter-reference (s_1, s_2) is

$$\frac{\mathrm{w}(s_1)\mathrm{w}(s_2)}{\sum_{(v_1,v_2)\in E:\, v_1 \neq v_2} \mathrm{w}(v_1)\mathrm{w}(v_2)} \Big/ \frac{\mathrm{f}(s_1, s_2)}{\sum_{(v_1,v_2)\in E:\, v_1 \neq v_2} \mathrm{f}(v_1, v_2)} \,.$$

For a fixed reference graph, the denominators of the two fractions (i.e. the total d-ref coupling and the total size of the inter-references) are constant, and the d-ref leverage is proportional to $\frac{\mathrm{w}(s_1)\mathrm{w}(s_2)}{\mathrm{f}(s_1,s_2)}$, i.e. the inverse inter-density from s_1 to s_2. Thus references with large d-ref leverage are *sparse* references.

11.3.2 Visualization

In matrix visualizations of a reference graph, each inter-reference corresponds to a non-white matrix element outside the diagonal. The d-ref leverage of a reference is inversely proportional to its density, which corresponds to the color density of the matrix element. Thus inter-references with large d-ref leverage correspond to matrix elements outside the diagonal with small but non-zero color density.

In the visualization of JHotDraw in Figure 11.6 on page 196, the two marked matrix elements correspond to inter-references with large d-ref leverage, while the inter-references to the subsystem framework have small d-ref leverage. A certain minimum color density is always enforced for all non-white matrix elements, to ensure that they are recognizable as non-white.

11.3.3 Relation to Design Flaws

An experiment was conducted to evaluate the following two hypotheses:

- Software systems contain system parts with a d-ref leverage much greater than 1; in other words, small parts of the systems cause a large part of their total d-ref coupling.
- The d-ref leverage indicates design flaws with high precision, both in absolute terms and relative to random selection.

Precision is a common measure for the quality of search results in information retrieval, and is often used together with recall (e.g. [vR79, Chapter 7]). For a given set of search results and a given set of design flaws, the fraction of flaws among the search results is called *precision*, and the fraction of search results among the flaws is called *recall*. If system parts with high d-ref leverage are mostly design flaws, then the total d-ref coupling is an indicator of design quality (as defined in Subsection 9.1.2).

Method The analyzed software systems ArgoUML 0.22, Eclipse 3.1, JDK 1.4.2, JHotDraw 5.4, and JWAM 1.8 are described in Appendix A.3.

To examine the existence of inter-references with large d-ref leverage, the d-ref leverage of all inter-references was computed, and the inter-references of each system were ordered by decreasing d-ref leverage.

To examine the precision of the d-ref leverage as indicator of design flaws, each inter-reference was manually classified as flawed or non-flawed, based on the responsibilities of the subsystems; see Appendix A.3 for details. The precision and recall were computed for the most leveraged inter-reference, the two most leveraged inter-references, etc., of each software system. For comparison, the expected precision for sets of randomly selected inter-references was determined; it equals the fraction of flawed references among all inter-references of the system.

Results Matrix visualizations of the software systems are shown in Figure 11.3 to Figure 11.7, and the inter-references are listed in Table 11.2 to Table 11.6, both on pages 190 to 197.

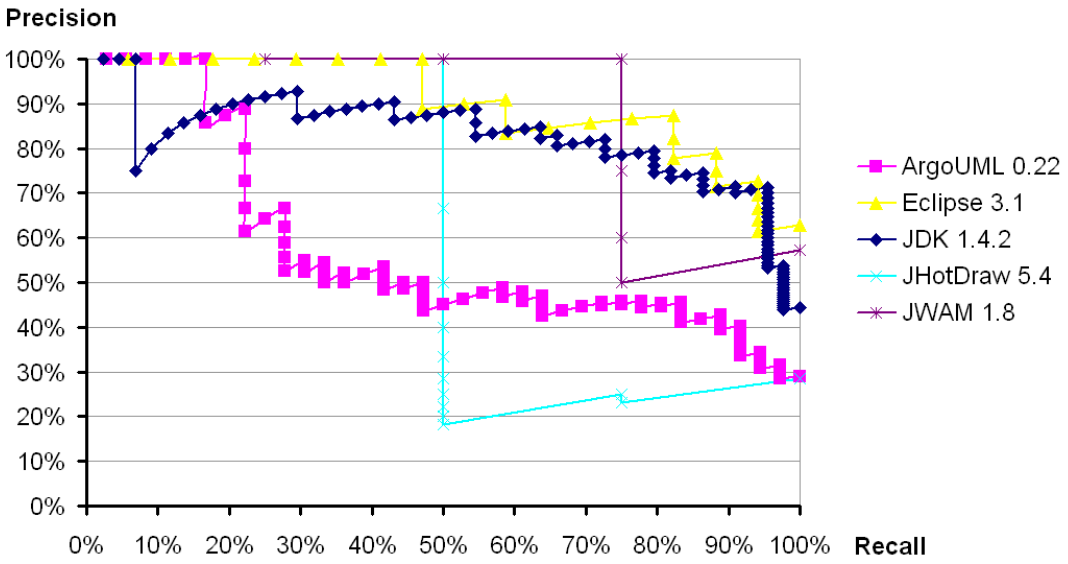
The tables show that all five software systems contain inter-references with a d-ref leverage significantly greater than 1. The largest d-ref leverage is between 34.9 (for JWAM) and 608.2 (for JDK). Only 0.02% (JDK) to 0.56% (JWAM) of the inter-references cause 10% of the total d-ref coupling, and only 0.26% (JDK) to 6.88% (JWAM) of the inter-references cause 25% of the total d-ref coupling.

The precision and recall of the d-ref leverage as indicator of flawed inter-references are summarized in Figure 11.2a. Halve of all flawed references are recalled with at least 88% precision for all systems except ArgoUML, where the precision is 45%. Even 75% of all flawed references are recalled with at least 78% precision for Eclipse, JDK, and JWAM. For all systems, the precision of the d-ref leverage is much higher than the precision of random selection, which is shown in Figure 11.2c.

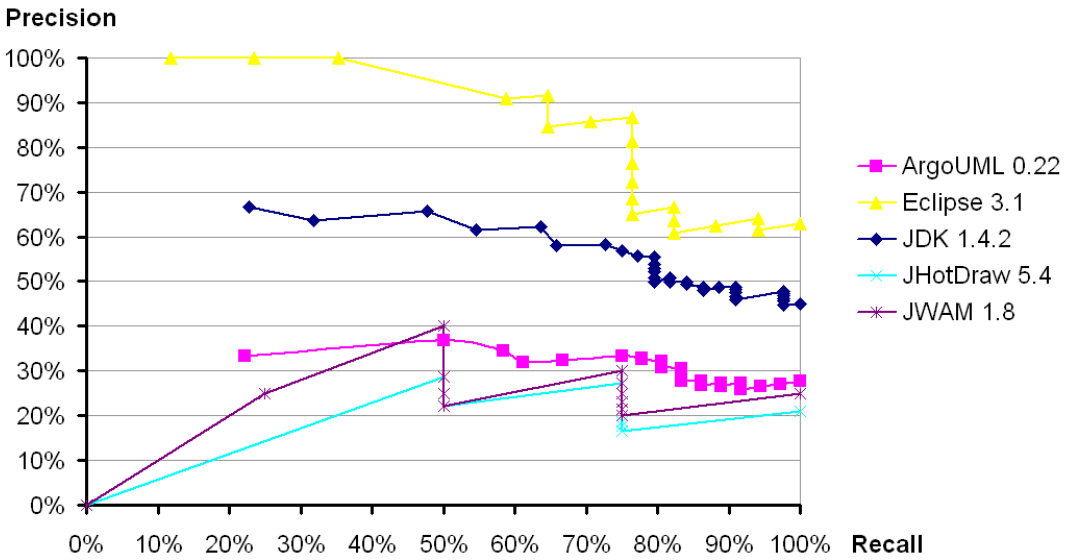
Discussion The results clearly support both hypotheses, namely the presence of system parts with large d-ref leverage in real-world software systems, and the validity of the d-ref leverage as indicator of design flaws (which implies the validity of the total d-ref coupling as indicator of design quality).

The results have been affected by the following parameters: the examined software systems, the chosen subsystem decompositions, the type of system part for which the d-ref leverage was computed, and the authoritative classification of the system parts into flaws and non-flaws (only for the second hypothesis). The values of these parameters have been identical for the two compared indicators of design flaws, namely d-ref leverage and random selection; thus the observed difference in the precision of the indicators was indeed caused by the indicators. However, it needs to be discussed to what degree the results can be generalized to situations with other parameter values.

The examined *software systems* were of diverse sizes (300 to 11 645 software elements), but otherwise fairly similar: They are all object-oriented systems written in Java, are entirely or mostly libraries or frameworks (except ArgoUML), are used by a large number of developers or end users (except probably JWAM), and their overall design quality is arguable fairly good. It is open whether the results generalize to other programming languages and to less well-designed systems.



(a) Inter-references with largest d-ref leverage (Table 11.2 to 11.6)



(b) Inter-references with smallest size (Subsection 11.3.4)

System	Expected Precision
ArgoUML 0.22	36 / 148 = 24.3%
Eclipse 3.1	17 / 113 = 15.0%
JDK 1.4.2	44 / 130 = 33.8%
JHotDraw 5.4	4 / 30 = 13.3%
JWAM 1.8	4 / 32 = 12.5%

(c) Randomly selected inter-references

Figure 11.2: Precision and recall of three methods for detecting flawed references

Nr	Reference	Size of Reference		Caused Coupling		Leverage		Flaw
		This	Total	This	Total	This	Total	
1	(uml.ui.foundation, uml.ui.behavior)	1 (0.02%)	0.02%	34293 (3.75%)	3.75%	186.6	186.6	yes
2	(ui, uml.cognitive)	1 (0.02%)	0.04%	23205 (2.54%)	6.29%	126.3	156.5	yes
3	(uml.ui.foundation, uml.diagram)	2 (0.04%)	0.08%	38318 (4.19%)	10.5%	104.3	130.4	yes
4	(uml.ui.behavior, uml.diagram)	3 (0.06%)	0.14%	50694 (5.54%)	16.0%	92.0	113.9	yes
5	(cognitive, uml.cognitive)	1 (0.02%)	0.16%	8568 (0.94%)	17.0%	46.6	105.5	yes
6	(ui, uml.notation)	1 (0.02%)	0.18%	8385 (0.92%)	17.9%	45.6	98.9	yes
7	(uml.cognitive, ui)	5 (0.10%)	0.28%	23205 (2.54%)	20.4%	25.3	72.6	no
8	(uml.diagram, uml.reveng)	1 (0.02%)	0.30%	4284 (0.47%)	20.9%	23.3	69.3	yes
9	(cognitive, uml.ui.ui)	2 (0.04%)	0.34%	7920 (0.87%)	21.7%	21.6	63.7	yes
10	(uml.diagram, util)	1 (0.02%)	0.36%	3808 (0.42%)	22.2%	20.7	61.3	no
11	(uml.diagram, uml.notation)	3 (0.06%)	0.42%	10234 (1.12%)	23.3%	18.6	55.2	no
12	(uml.diagram, uml.uml)	1 (0.02%)	0.44%	3094 (0.34%)	23.6%	16.8	53.4	no
13	(persistence, ui)	3 (0.06%)	0.50%	7410 (0.81%)	24.4%	13.4	48.6	no
14	(uml.ui.ui, language)	1 (0.02%)	0.52%	2420 (0.26%)	24.7%	13.2	47.3	yes
15	(uml.cognitive, persistence)	2 (0.04%)	0.56%	4522 (0.49%)	25.2%	12.3	44.8	yes
16	(uml.diagram, uml.ui.ui)	12 (0.24%)	0.80%	26180 (2.86%)	28.0%	11.9	34.9	no
17	(uml.generator, ui)	1 (0.02%)	0.82%	2145 (0.23%)	28.3%	11.7	34.3	no
18	(uml.reveng, uml.ui.ui)	1 (0.02%)	0.84%	1980 (0.22%)	28.5%	10.8	33.8	no
19	(uml.diagram, cognitive)	10 (0.20%)	1.04%	17136 (1.87%)	30.4%	9.33	29.1	no
20	(uml.uml, uml.diagram)	2 (0.04%)	1.08%	3094 (0.34%)	30.7%	8.42	28.3	yes
21	(uml.ui.behavior, ui)	29 (0.58%)	1.67%	41535 (4.54%)	35.3%	7.80	21.1	no
22	(uml.cognitive, language)	2 (0.04%)	1.71%	2618 (0.29%)	35.5%	7.12	20.8	yes
23	(uml.reveng, cognitive)	1 (0.02%)	1.73%	1296 (0.14%)	35.7%	7.05	20.7	no
24	(uml.ui.foundation, ui)	26 (0.52%)	2.25%	31395 (3.43%)	39.1%	6.57	17.4	no
25	(ui, uml.diagram)	41 (0.82%)	3.07%	46410 (5.07%)	44.2%	6.16	14.4	yes
26	(persistence, uml.cognitive)	4 (0.08%)	3.15%	4522 (0.49%)	44.7%	6.15	14.2	no
27	(uml.ui.ui, persistence)	4 (0.08%)	3.23%	4180 (0.46%)	45.1%	5.69	14.0	yes
28	(uml.ui.ui, uml.reveng)	2 (0.04%)	3.27%	1980 (0.22%)	45.4%	5.39	13.9	yes
29	(notation, ui)	2 (0.04%)	3.31%	1755 (0.19%)	45.5%	4.78	13.7	no
30	(uml.ui.ui, uml.uml)	2 (0.04%)	3.35%	1430 (0.16%)	45.7%	3.89	13.6	no
31	(uml.ui.model_management, ui)	2 (0.04%)	3.39%	1365 (0.15%)	45.9%	3.71	13.5	no
32	(uml.ui.ui, uml.ui.foundation)	27 (0.54%)	3.94%	17710 (1.94%)	47.8%	3.57	12.1	yes
33	(uml.cognitive, uml.diagram)	45 (0.90%)	4.84%	28322 (3.10%)	50.9%	3.43	10.5	no
34	(ui, persistence)	12 (0.24%)	5.08%	7410 (0.81%)	51.7%	3.36	10.2	yes
35	(uml.ui.behavior, uml.ui.foundation)	56 (1.12%)	6.21%	34293 (3.75%)	55.4%	3.33	8.93	no
36	(cognitive, util)	2 (0.04%)	6.25%	1152 (0.13%)	55.6%	3.13	8.89	no
37	(cognitive, model)	11 (0.22%)	6.47%	6048 (0.66%)	56.2%	2.99	8.69	no
38	(language, uml.cognitive)	5 (0.10%)	6.57%	2618 (0.29%)	56.5%	2.85	8.60	no
39	(cognitive, ui)	28 (0.56%)	7.13%	14040 (1.54%)	58.1%	2.73	8.14	no
40	(uml.ui.ui, uml.diagram)	53 (1.06%)	8.20%	26180 (2.86%)	60.9%	2.69	7.43	yes
41	(uml.ui.ui, uml.ui.behavior)	48 (0.96%)	9.16%	23430 (2.56%)	63.5%	2.66	6.93	yes
42	(cognitive, uml.uml)	2 (0.04%)	9.20%	936 (0.10%)	63.6%	2.55	6.91	yes
43	(ui, cognitive)	30 (0.60%)	9.80%	14040 (1.54%)	65.1%	2.55	6.64	yes
44	(ui, util)	7 (0.14%)	9.94%	3120 (0.34%)	65.5%	2.43	6.58	no
45	(uml.ui.ui, util)	4 (0.08%)	10.0%	1760 (0.19%)	65.6%	2.39	6.55	no
46	(ui, uml.ui.ui)	49 (0.98%)	11.0%	21450 (2.35%)	68.0%	2.38	6.18	yes
47	(persistence, uml.diagram)	21 (0.42%)	11.4%	9044 (0.99%)	69.0%	2.34	6.04	no
48	(uml.diagram, ui)	108 (2.17%)	13.6%	46410 (5.07%)	74.1%	2.34	5.45	no
49	(ui, uml.uml)	6 (0.12%)	13.7%	2535 (0.28%)	74.3%	2.30	5.42	yes
50	(language, cognitive)	4 (0.08%)	13.8%	1584 (0.17%)	74.5%	2.16	5.40	no
51	(uml.reveng, ui)	9 (0.18%)	14.0%	3510 (0.38%)	74.9%	2.12	5.36	no
52	(moduleloader, ui)	2 (0.04%)	14.0%	780 (0.09%)	75.0%	2.12	5.35	no
53	(uml.cognitive, uml.uml)	4 (0.08%)	14.1%	1547 (0.17%)	75.1%	2.10	5.33	no
54	(uml.cognitive, ocl)	2 (0.04%)	14.1%	714 (0.08%)	75.2%	1.94	5.32	no
55	(kernel, uml.cognitive)	3 (0.06%)	14.2%	1071 (0.12%)	75.3%	1.94	5.30	yes
56	(ui, notation)	5 (0.10%)	14.3%	1755 (0.19%)	75.5%	1.91	5.28	yes
57	(kernel, persistence)	1 (0.02%)	14.3%	342 (0.04%)	75.6%	1.86	5.28	yes
58	(uml.ui.ui, ocl)	2 (0.04%)	14.4%	660 (0.07%)	75.6%	1.80	5.27	no
59	(kernel, cognitive)	2 (0.04%)	14.4%	648 (0.07%)	75.7%	1.76	5.26	yes
60	(persistence, cognitive)	9 (0.18%)	14.6%	2736 (0.30%)	76.0%	1.65	5.21	no
61	(kernel, ui)	6 (0.12%)	14.7%	1755 (0.19%)	76.2%	1.59	5.18	yes
...	...	...	...	...	...	...	...	...
64	(uml.ui.ui, uml.ui.model_management)	3 (0.06%)	14.8%	770 (0.08%)	76.4%	1.40	5.15	yes
66	(kernel, uml.diagram)	9 (0.18%)	15.7%	2142 (0.23%)	77.5%	1.30	4.94	yes
74	(notation, uml.notation)	2 (0.04%)	18.4%	387 (0.04%)	80.9%	1.05	4.40	yes
75	(uml.ui.ui, uml.generator)	7 (0.14%)	18.5%	1210 (0.13%)	81.0%	0.94	4.37	yes
82	(kernel, uml.generator)	1 (0.02%)	18.9%	99 (0.01%)	81.2%	0.54	4.31	yes
99	(pattern, uml.cognitive)	6 (0.12%)	37.3%	357 (0.04%)	89.1%	0.32	2.39	yes
111	(kernel, notation)	2 (0.04%)	58.3%	81 (0.01%)	94.2%	0.22	1.62	yes
124	(kernel, uml.uml)	5 (0.10%)	88.2%	117 (0.01%)	99.6%	0.13	1.13	yes
148	(uml.ui.model_management, i18n)	6 (0.12%)	100%	7 (0.00%)	100%	0.01	1.00	no

Table 11.2: Inter-references in ArgoUML 0.22, ordered by d-ref leverage

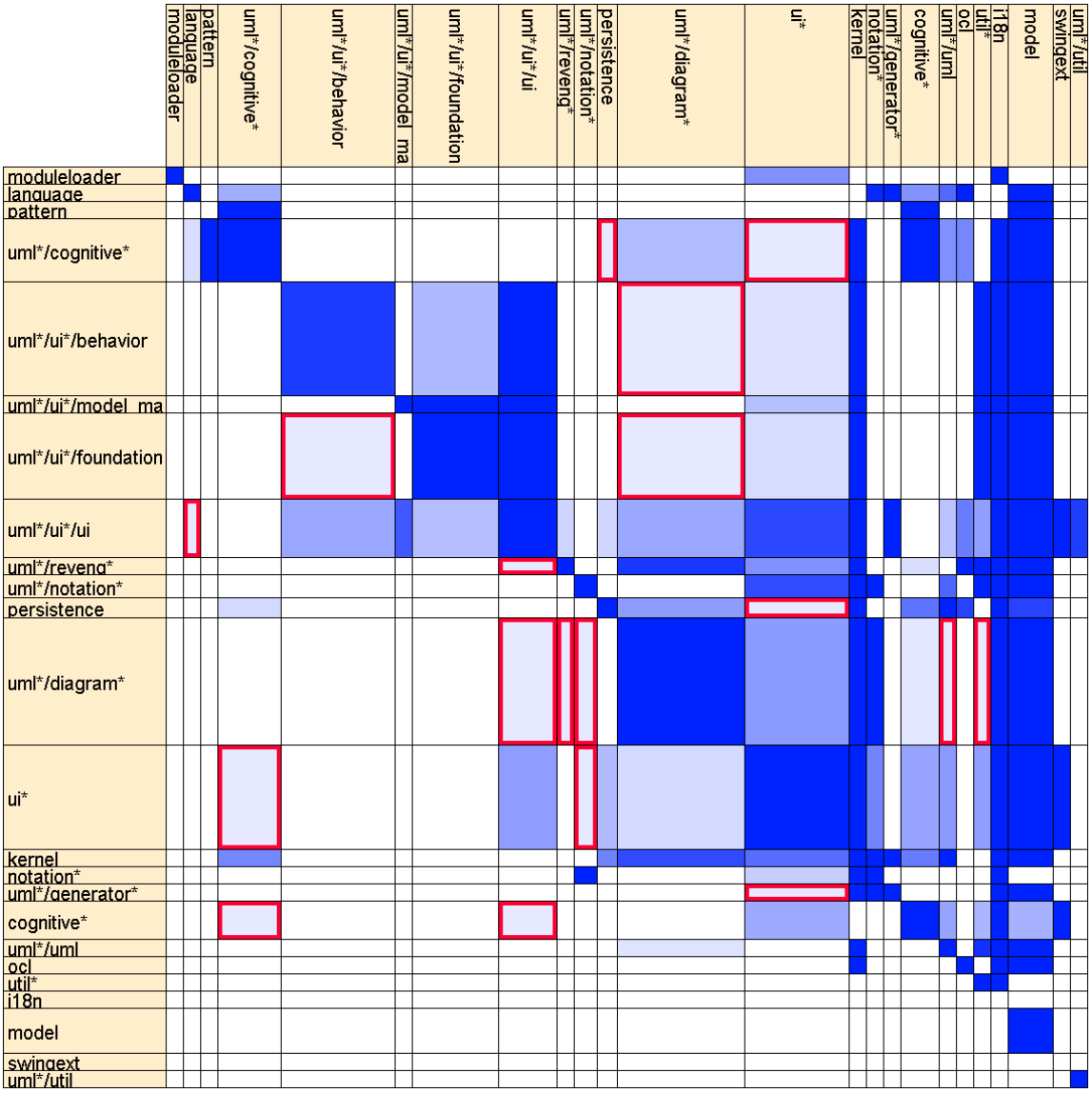


Figure 11.3: D-ref dependencies in ArgoUML 0.22. Removing the marked 0.84% of all inter-references (number 1 to 18 in Table 11.2) reduces the d-ref coupling by 28.5% from 914 628 (total) or 0.525 (scaled) to 653 963 or 0.375.

Nr	Reference	Size of Reference		Caused Coupling		Leverage		Flaw
		This	Total	This	Total	This	Tot	
1	(JDT UI, Team)	1 (0.00%)	0.00%	488704 (0.89%)	0.89%	554	554	yes
2	(PDE UI, Help)	1 (0.00%)	0.00%	246016 (0.45%)	1.34%	279	416	yes
3	(Ant, JDT UI)	4 (0.01%)	0.01%	723511 (1.32%)	2.66%	205	275	yes
4	(Runtime, Workbench)	11 (0.02%)	0.03%	1280832 (2.33%)	4.99%	132	182	yes
5	(Workbench, Update)	7 (0.01%)	0.04%	693784 (1.26%)	6.26%	112	162	yes
6	(Runtime, SWT)	4 (0.01%)	0.05%	325248 (0.59%)	6.85%	92.2	152	yes
7	(Debug, Help)	2 (0.00%)	0.05%	157696 (0.29%)	7.14%	89.4	148	yes
8	(JDT UI, Help)	7 (0.01%)	0.06%	488704 (0.89%)	8.03%	79.2	135	yes
9	(PDE UI, JDT UI)	33 (0.05%)	0.11%	1834549 (3.34%)	11.4%	63.0	101	no
10	(PDE UI, Ant)	7 (0.01%)	0.12%	364219 (0.66%)	12.0%	59.0	97.2	yes
11	(PDE UI, Team)	7 (0.01%)	0.14%	246016 (0.45%)	12.5%	39.9	92.5	yes
12	(PDE UI, Update)	10 (0.02%)	0.15%	349804 (0.64%)	13.1%	39.7	86.8	no
13	(Runtime, Update)	8 (0.01%)	0.16%	244608 (0.45%)	13.6%	34.7	82.7	yes
14	(Workbench, Help)	17 (0.03%)	0.19%	487936 (0.89%)	14.5%	32.5	75.6	yes
15	(JDT UI, JDT Debug)	58 (0.09%)	0.28%	1597833 (2.91%)	17.4%	31.2	61.0	yes
16	(Workbench, Text)	54 (0.09%)	0.37%	1206498 (2.20%)	19.6%	25.3	52.7	yes
17	(PDE UI, JDT Debug)	49 (0.08%)	0.45%	804357 (1.47%)	21.0%	18.6	46.7	no
18	(JDT UI, Debug)	72 (0.12%)	0.57%	1175944 (2.14%)	23.2%	18.5	41.0	no
19	(PDE Build, Ant)	2 (0.00%)	0.57%	26530 (0.05%)	23.2%	15.0	40.8	yes
20	(PDE UI, PDE Build)	7 (0.01%)	0.58%	67270 (0.12%)	23.3%	10.9	40.2	no
21	(CVS, Text)	30 (0.05%)	0.63%	288015 (0.52%)	23.9%	10.9	38.0	no
22	(Ant, JDT Core)	42 (0.07%)	0.70%	396055 (0.72%)	24.6%	10.7	35.3	yes
23	(Update, Workbench)	89 (0.14%)	0.84%	693784 (1.26%)	25.9%	8.84	30.8	no
24	(JDT Debug, JDT UI)	227 (0.36%)	1.20%	1597833 (2.91%)	28.8%	7.98	23.9	no
25	(JDT Core, Text)	120 (0.19%)	1.40%	661485 (1.21%)	30.0%	6.25	21.5	no
26	(PDE UI, Debug)	111 (0.18%)	1.58%	591976 (1.08%)	31.1%	6.05	19.7	no
27	(Ant, JDT Debug)	65 (0.10%)	1.68%	317223 (0.58%)	31.6%	5.53	18.8	yes
28	(Debug, Text)	80 (0.13%)	1.81%	389928 (0.71%)	32.3%	5.53	17.9	no
29	(Text, Workbench)	266 (0.43%)	2.24%	1206498 (2.20%)	34.5%	5.14	15.4	no
30	(PDE UI, JDT Core)	258 (0.41%)	2.65%	1004245 (1.83%)	36.4%	4.41	13.7	no
31	(PDE UI, Search)	34 (0.05%)	2.71%	129735 (0.24%)	36.6%	4.33	13.5	no
32	(JDT Debug,Workbench)	474 (0.76%)	3.47%	1595322 (2.91%)	39.5%	3.82	11.4	no
33	(JDT UI, Search)	80 (0.13%)	3.60%	257715 (0.47%)	40.0%	3.65	11.1	no
34	(JFace, Runtime)	64 (0.10%)	3.70%	176736 (0.32%)	40.3%	3.13	10.9	no
35	(Ant, Workbench)	273 (0.44%)	4.14%	722374 (1.32%)	41.6%	3.00	10.1	no
36	(JDT Debug, Text)	219 (0.35%)	4.49%	529821 (0.97%)	42.6%	2.74	9.49	no
37	(Compare, Workbench)	95 (0.15%)	4.64%	223002 (0.41%)	43.0%	2.66	9.26	no
38	(Help, Workbench)	218 (0.35%)	4.99%	487936 (0.89%)	43.9%	2.54	8.79	no
39	(PDE UI, Text)	272 (0.44%)	5.43%	608313 (1.11%)	45.0%	2.54	8.29	no
40	(JDT UI, Compare)	108 (0.17%)	5.60%	223353 (0.41%)	45.4%	2.34	8.10	no
41	(Compare, Text)	38 (0.06%)	5.66%	74061 (0.13%)	45.5%	2.21	8.04	no
42	(CVS, Workbench)	488 (0.78%)	6.45%	867230 (1.58%)	47.1%	2.02	7.31	no
43	(JDT Core, Runtime)	409 (0.66%)	7.11%	702240 (1.28%)	48.4%	1.95	6.81	no
44	(Text, Workspace)	113 (0.18%)	7.29%	181671 (0.33%)	48.7%	1.82	6.69	no
45	(Team, Workbench)	305 (0.49%)	7.78%	487936 (0.89%)	49.6%	1.81	6.38	no
...	...	...	...	...	...	...	...	...
113	(Search, JFace)	307 (0.49%)	100%	35505 (0.06%)	100%	0.13	1.00	no

Table 11.3: Inter-references in Eclipse 3.1, ordered by d-ref leverage

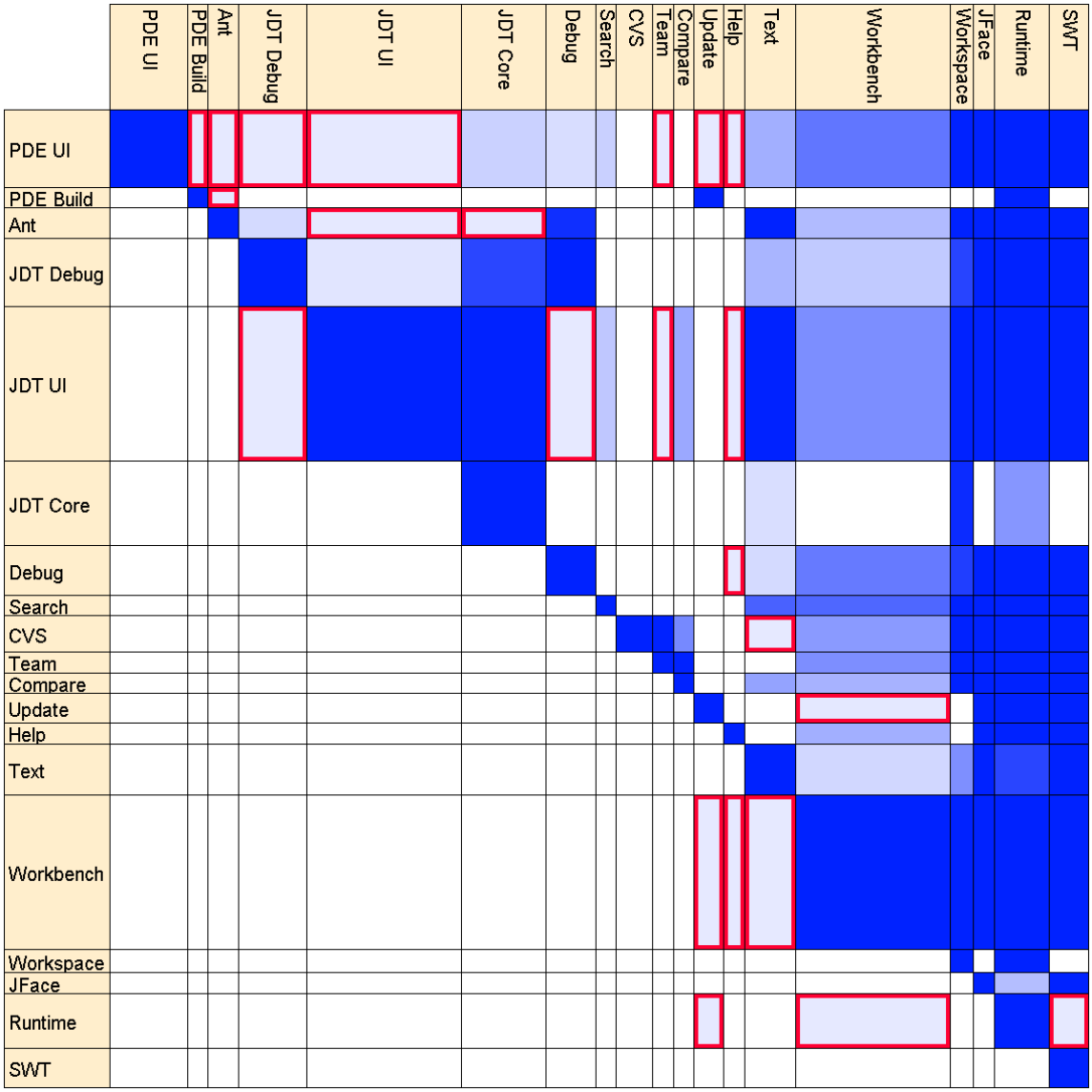


Figure 11.4: D-ref dependencies in Eclipse 3.1. Removing the marked 0.84% of all inter-references (number 1 to 23 in Table 11.3) reduces the d-ref coupling by 25.9% from 54.9 million (total) or 0.445 (scaled) to 40.7 million or 0.330.

Nr	Reference	Size of Reference		Caused Coupling		Leverage		Flaw
		This	Total	This	Total	This	Total	
1	(java.awt, javax.swing)	2 (0.02%)	0.02%	197532 (10.7%)	10.7%	608.2	608.2	yes
2	(java.awt, java.rmi)	1 (0.01%)	0.03%	23364 (1.27%)	12.0%	143.9	453.5	yes
3	(java.awt, java.nio)	2 (0.02%)	0.04%	46728 (2.54%)	14.6%	143.9	329.6	no
4	(javax.print, java.awt)	3 (0.03%)	0.07%	43188 (2.35%)	16.9%	88.7	239.3	yes
5	(javax.accessibility, javax.swing)	1 (0.01%)	0.08%	12834 (0.70%)	17.6%	79.0	221.5	yes
6	(java.awt, javax.print)	5 (0.04%)	0.12%	43188 (2.35%)	20.0%	53.2	161.4	yes
7	(javax.imageio, java.security)	1 (0.01%)	0.13%	8322 (0.45%)	20.4%	51.3	154.0	yes
8	(java.text, java.awt)	2 (0.02%)	0.15%	16284 (0.89%)	21.3%	50.1	141.8	yes
9	(javax.sound, java.security)	1 (0.01%)	0.16%	7884 (0.43%)	21.7%	48.6	136.6	yes
10	(java.util, java.net)	1 (0.01%)	0.17%	7560 (0.41%)	22.1%	46.6	131.9	yes
11	(javax.naming, java.security)	2 (0.02%)	0.19%	14016 (0.76%)	22.9%	43.2	123.4	yes
12	(javax.naming, java.net)	1 (0.01%)	0.19%	5760 (0.31%)	23.2%	35.5	119.4	yes
13	(java.util, java.nio)	3 (0.03%)	0.22%	16632 (0.90%)	24.1%	34.1	109.2	yes
14	(java.awt, java.net)	4 (0.04%)	0.26%	21240 (1.16%)	25.3%	32.7	98.6	yes
15	(javax.swing, javax.sound)	6 (0.05%)	0.31%	30132 (1.64%)	26.9%	30.9	87.0	no
16	(java.nio, java.net)	2 (0.02%)	0.33%	7920 (0.43%)	27.3%	24.4	83.7	yes
17	(javax.imageio, java.net)	1 (0.01%)	0.34%	3420 (0.19%)	27.5%	21.1	82.0	yes
18	(java.security, java.net)	3 (0.03%)	0.36%	8760 (0.48%)	28.0%	18.0	77.3	yes
19	(javax.swing, java.security)	29 (0.26%)	0.62%	81468 (4.43%)	32.4%	17.3	52.5	yes
20	(java.text, java.net)	1 (0.01%)	0.63%	2760 (0.15%)	32.6%	17.0	52.0	yes
21	(javax.print, java.security)	7 (0.06%)	0.69%	17812 (0.97%)	33.6%	15.7	48.7	yes
22	(javax.imageio, java.nio)	3 (0.03%)	0.72%	7524 (0.41%)	34.0%	15.4	47.5	no
23	(java.nio, java.security)	8 (0.07%)	0.79%	19272 (1.05%)	35.0%	14.8	44.5	yes
24	(java.text, java.security)	3 (0.03%)	0.81%	6716 (0.37%)	35.4%	13.8	43.5	yes
25	(javax.security, java.net)	1 (0.01%)	0.82%	2220 (0.12%)	35.5%	13.7	43.2	yes
26	(java.sql, java.security)	2 (0.02%)	0.84%	4380 (0.24%)	35.7%	13.5	42.6	yes
27	(java.awt, java.security)	26 (0.23%)	1.07%	51684 (2.81%)	38.6%	12.2	36.1	yes
28	(javax.accessibility, java.io)	1 (0.01%)	1.08%	1840 (0.10%)	38.7%	11.3	35.9	no
29	(java.awt, java.math)	1 (0.01%)	1.09%	1770 (0.10%)	38.8%	10.9	35.7	no
30	(java.io, java.nio)	7 (0.06%)	1.15%	10560 (0.57%)	39.3%	9.29	34.2	yes
31	(javax.print, java.net)	5 (0.04%)	1.19%	7320 (0.40%)	39.7%	9.02	33.3	yes
32	(java.lang, java.nio)	10 (0.09%)	1.28%	14520 (0.79%)	40.5%	8.94	31.6	yes
33	(java.awt, java.applet)	1 (0.01%)	1.29%	1416 (0.08%)	40.6%	8.72	31.5	yes
34	(java.rmi, java.util)	6 (0.05%)	1.34%	8316 (0.45%)	41.0%	8.54	30.6	no
35	(java.lang, java.net)	5 (0.04%)	1.39%	6600 (0.36%)	41.4%	8.13	29.8	yes
36	(javax.xml, java.util)	3 (0.03%)	1.41%	3780 (0.21%)	41.6%	7.76	29.4	no
37	(java.util, javax.xml)	3 (0.03%)	1.44%	3780 (0.21%)	41.8%	7.76	29.0	yes
38	(java.io, java.net)	4 (0.04%)	1.48%	4800 (0.26%)	42.1%	7.39	28.5	yes
39	(javax.swing, java.net)	28 (0.25%)	1.72%	33480 (1.82%)	43.9%	7.36	25.5	yes
40	(java.nio, java.util)	14 (0.12%)	1.85%	16632 (0.90%)	44.8%	7.32	24.3	no
41	(java.net, java.nio)	7 (0.06%)	1.91%	7920 (0.43%)	45.2%	6.97	23.7	no
42	(javax.accessibility, java.awt)	9 (0.08%)	1.99%	8142 (0.44%)	45.7%	5.57	23.0	yes
43	(java.security, javax.security)	6 (0.05%)	2.04%	5402 (0.29%)	46.0%	5.54	22.5	yes
44	(java.util, java.text)	7 (0.06%)	2.10%	5796 (0.32%)	46.3%	5.10	22.0	yes
45	(javax.sound, java.util)	9 (0.08%)	2.18%	6804 (0.37%)	46.7%	4.66	21.4	no
46	(java.rmi, java.security)	13 (0.11%)	2.30%	9636 (0.52%)	47.2%	4.56	20.5	no
47	(java.awt, java.text)	23 (0.20%)	2.50%	16284 (0.89%)	48.1%	4.36	19.2	no
48	(javax.sound, java.net)	5 (0.04%)	2.54%	3240 (0.18%)	48.2%	3.99	19.0	yes
49	(javax.imageio, java.awt)	32 (0.28%)	2.83%	20178 (1.10%)	49.3%	3.88	17.5	no
50	(java.math, java.util)	1 (0.01%)	2.84%	630 (0.03%)	49.4%	3.88	17.4	no
51	(java.io, java.security)	19 (0.17%)	3.00%	11680 (0.64%)	50.0%	3.79	16.6	yes
52	(java.sql, java.net)	3 (0.03%)	3.03%	1800 (0.10%)	50.1%	3.70	16.5	yes
53	(javax.security, java.text)	3 (0.03%)	3.06%	1702 (0.09%)	50.2%	3.49	16.4	no
54	(java.applet, java.util)	1 (0.01%)	3.07%	504 (0.03%)	50.2%	3.10	16.4	no
55	(java.util, java.security)	37 (0.33%)	3.39%	18396 (1.00%)	51.2%	3.06	15.1	yes
56	(java.io, java.util)	21 (0.19%)	3.58%	10080 (0.55%)	51.8%	2.96	14.5	yes
57	(javax.swing, java.text)	55 (0.49%)	4.06%	25668 (1.40%)	53.2%	2.87	13.1	no
58	(java.lang, java.security)	37 (0.33%)	4.39%	16060 (0.87%)	54.1%	2.67	12.3	yes
59	(java.nio, java.io)	26 (0.23%)	4.62%	10560 (0.57%)	54.6%	2.50	11.8	no
60	(java.lang, java.util)	37 (0.33%)	4.95%	13860 (0.75%)	55.4%	2.31	11.2	yes
...	...	...	...	...	...	...	...	...
80	(java.lang, java.io)	48 (0.42%)	10.6%	8800 (0.48%)	63.9%	1.13	6.00	yes
99	(java.applet, java.net)	4 (0.04%)	42.0%	320 (0.02%)	86.1%	0.49	2.05	yes
130	(java.math, java.lang)	34 (0.30%)	100%	550 (0.03%)	100%	0.10	1.00	no

Table 11.4: Inter-references in JDK 1.4.2, ordered by d-ref leverage

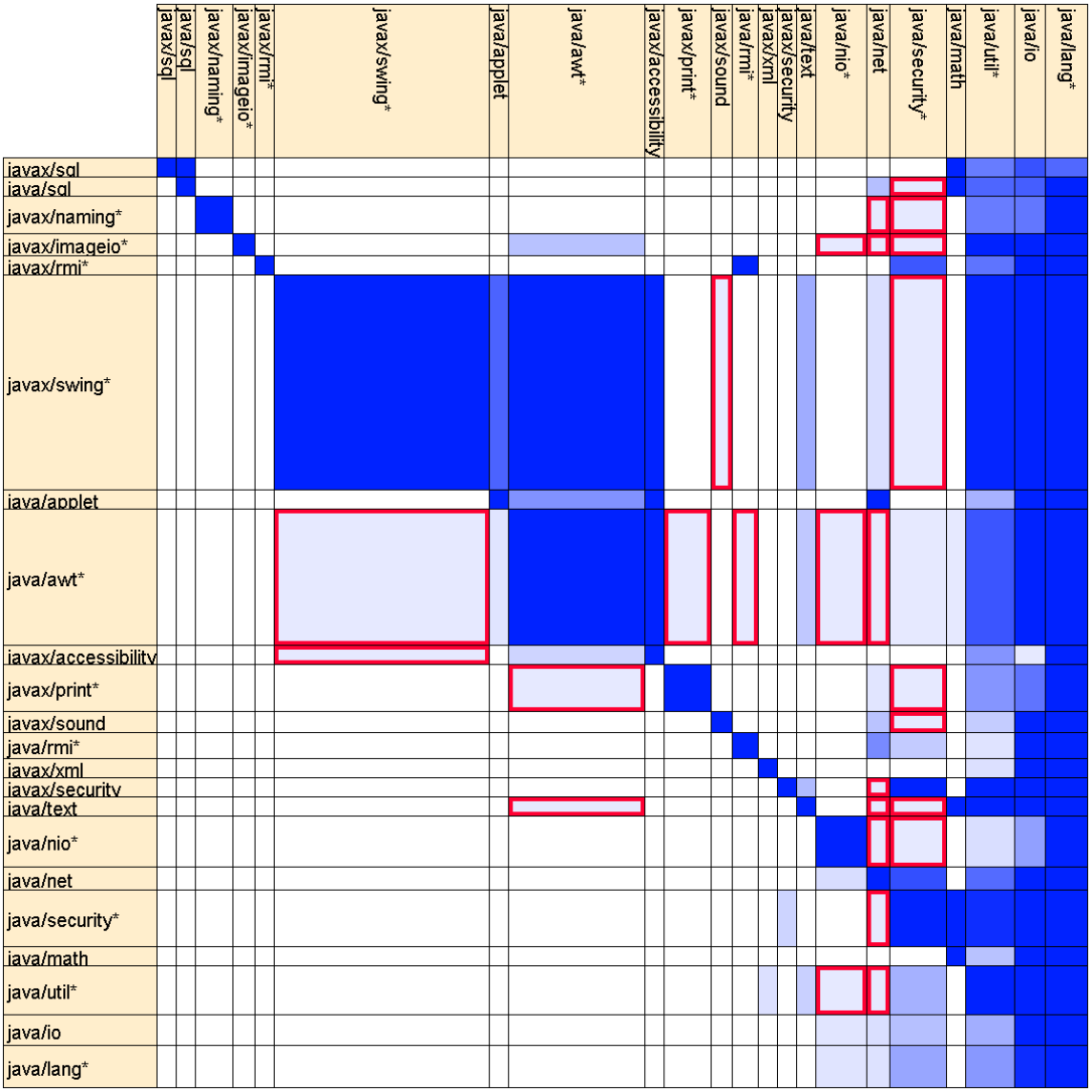


Figure 11.5: D-ref dependencies in JDK 1.4.2. Removing the marked 0.84% of all inter-references (number 1 to 26 in Table 11.4) reduces the d-ref coupling by 35.7% from 1 837 792 (total) or 0.447 (scaled) to 1 180 876 or 0.287.

Nr	Reference	Size of Reference		Caused Coupling		Leverage		Flaw
		This	Total	This	Total	This	Total	
1	(standard, contrib)	4 (0.36%)	0.36%	6862 (14.5%)	14.5%	40.8	40.8	yes
2	(util, standard)	8 (0.71%)	1.07%	3504 (7.43%)	22.0%	10.4	20.6	yes
3	(contrib, figures)	13 (1.16%)	2.23%	3384 (7.17%)	29.1%	6.20	13.1	no
4	(framework, util)	6 (0.53%)	2.76%	1248 (2.65%)	31.8%	4.95	11.5	no
5	(contrib, standard)	57 (5.08%)	7.84%	6862 (14.5%)	46.3%	2.87	5.91	no
...	...	...	...	...	...	...	...	...
12	(util, framework)	32 (2.85%)	37.3%	1248 (2.65%)	81.1%	0.93	2.17	yes
...	...	...	...	...	...	...	...	...
14	(application, contrib)	4 (0.36%)	43.4%	94 (0.20%)	85.0%	0.56	1.96	yes
...	...	...	...	...	...	...	...	...
30	(application, util)	19 (1.69%)	100%	48 (0.10%)	100%	0.06	1.00	no

Table 11.5: Inter-references in JHotDraw 5.4, ordered by d-ref leverage

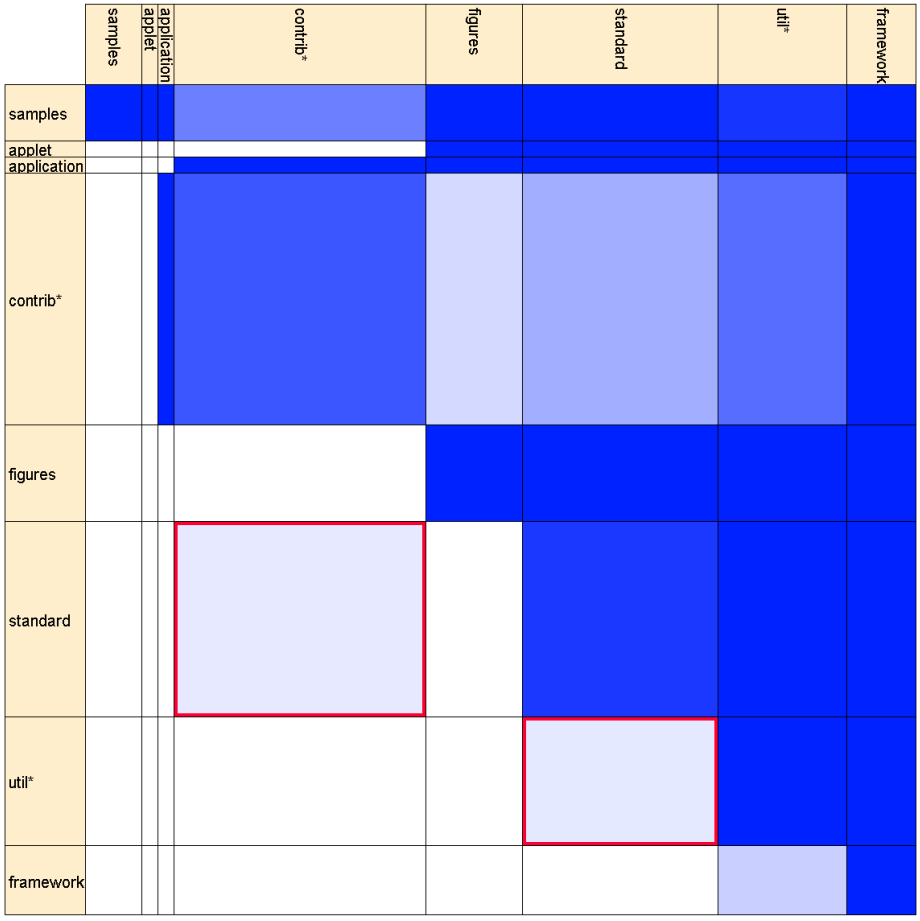


Figure 11.6: D-ref dependencies in JHotDraw 5.4. Removing the marked 1.07% of all inter-references (number 1 and 2 in Table 11.5) reduces the d-ref coupling by 22.0% from 47 171 (total) or 0.663 (scaled) to 36 805 or 0.518.

Nr	Reference	Size of Reference This	Total	Caused Coupling This	Total	Leverage This	Total	Flaw
1	(jwamx.handling, jwambeta.handling)	2 (0.10%)	0.10%	6208 (3.53%)	3.53%	34.9	34.9	yes
2	(jwamx.technology, jwam.handling)	9 (0.46%)	0.56%	14365 (8.18%)	11.7%	18.0	21.1	yes
3	(jwamdev, jwambeta.handling)	3 (0.15%)	0.71%	1664 (0.95%)	12.7%	6.24	17.9	yes
4	(jwamdev, jwam.handling)	6 (0.30%)	1.01%	2210 (1.26%)	13.9%	4.15	13.8	no
5	(jwamx.handling, jwamx.lang)	2 (0.10%)	1.11%	679 (0.39%)	14.3%	3.82	12.9	no
6	(jwambeta.handling, jwamx.handling)	22 (1.11%)	2.22%	6208 (3.53%)	17.8%	3.18	8.02	no
7	(jwamexample, jwambeta.handling)	31 (1.57%)	3.79%	8384 (4.77%)	22.6%	3.04	5.96	yes
8	(jwamx.technology, jwamx.lang)	5 (0.25%)	4.04%	1183 (0.67%)	23.3%	2.66	5.76	no
9	(jwam.technology, jwam.lang)	7 (0.35%)	4.40%	1591 (0.91%)	24.2%	2.56	5.50	no
10	(jwambeta.handling, jwamx.technology)	49 (2.48%)	6.88%	10816 (6.16%)	30.3%	2.48	4.41	no
...	...	...	...	...	...	...	...	...
32	(jwamx.lang, jwam.lang)	20 (1.01%)	100%	301 (0.17%)	100%	0.17	1.00	no

Table 11.6: Inter-references in JWAM 1.8, ordered by d-ref leverage

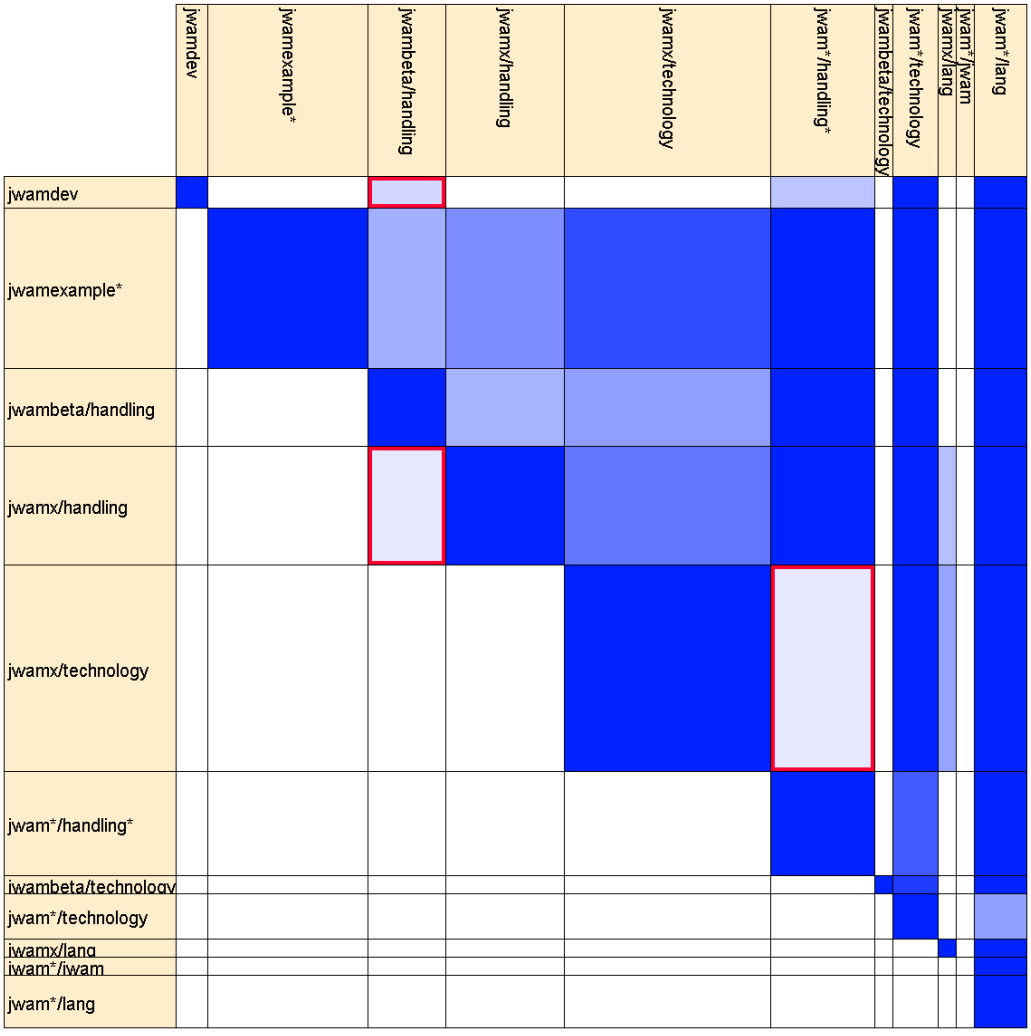


Figure 11.7: D-ref dependencies in JWAM 1.8. Removing the marked 0.71% of all inter-references (number 1 to 3 in Table 11.6) reduces the d-ref coupling by 12.7% from 175 711 (total) or 0.476 (scaled) to 153 474 or 0.416.

The chosen *subsystem decompositions* were coarse-grained. For fine-grained decompositions, the variance of the reference sizes is presumably smaller, because the largest references are smaller while the smallest references still have size 1. This could reduce the variance of the d-ref leverage, and its precision as flaw indicator.

The examined *system parts* were inter-subsystem references; the obvious alternatives are software elements or subsystems (see Subsection 11.3.1). References have been chosen because they are practically relevant (see Subsection 11.3.4), and enable a comparison with other indicators (see Subsections 11.3.4 and 12.2.4).

The *authoritative classification* of the references as flawed or non-flawed was derived partly by the author (see Appendix A.3). To avoid favorable effects on the precision of the d-ref leverage, the classification was derived without knowledge of the d-ref leverage, and clear criteria were applied.

Further Observations To complement the quantitative results in the previous paragraphs, this paragraph examine some failures of the d-ref leverage, i.e. inter-references with large d-ref leverage that are not design flaws (*false positives*), and inter-references with moderate d-ref leverage that are design flaws (*false negatives*).

False positives are non-flawed references where only a small part of the client subsystem refers to only a small part of the supplier subsystem. Such references occur if the client subsystem and/or the supplier subsystem are not cohesive, i.e. consist of software elements that do not reference similar subsystems or are not referenced by similar subsystems. For example, the subsystems cognitive, notation, uml.cognitive, uml.generator, and uml.reveng in ArgoUML implement some functionality with a small graphical user interface (GUI), and only their GUI part references the GUI subsystem ui. Similarly, the GUI framework javax.swing in JDK primarily uses graphics, and only a small part references the sound subsystem javax.sound. Another example are collections of loosely related utility classes which are not necessarily referenced together, like util, ui, uml.uml, and uml.ui.ui in ArgoUML, or java.io, java.util, java.nio, and java.text in JDK. A further potential cause of false positives, which apparently does not occur in the example systems, is the sparsification of non-flawed references through unified subsystem interfaces (facades [GHJV95]).

False negatives are flawed references that are nevertheless fairly dense, i.e. a significant part of the client subsystem refers to a significant part of the supplier subsystem. References from or to small subsystems (like kernel in ArgoUML, java.applet in JDK, and application in JHotDraw) are always fairly dense, and thus the density is not a reliable indicator of flaws for such references. Flawed but dense references between larger subsystems may result from continued design erosion, i.e. the introduction of more and more unintended references from a lower-level subsystem like ui and uml.ui.ui in ArgoUML or util in JHotDraw to higher-level subsystems by undisciplined or unknowing developers. Dense flawed references may also originate from subsystems that primarily have low-level responsibilities, but also provide some higher-level services; for example, the subsystem java.lang in JDK contains elementary classes e.g. for basic data types and exceptions, but also complex classes like ClassLoader and SecurityManager.

11.3.4 Related Work

This subsection discusses related work on the detection of flawed references, which often focuses on object-oriented programs. Specific tools are only mentioned if they contribute original indicators of design flaws; many further tools are available, in particular for computing reference-based software measures.

Manually Specified References When all acceptable references between subsystems are known, then the flawed references between subsystems can be easily computed as difference between the actually existing references and the acceptable references. The best-known technique of this type are the software reflexion models of Murphy et al. [MNS95, MNS01], but similar techniques [SSC96, MWD99, Pos03] and extensions [HH04b, CKS05] in particular to hierarchies of subsystems [KS03, OMB03, SJSJ05] have been proposed by many other authors. The implementation of these techniques in several free and commercial tools, e.g. Bauhaus [Axi, RVP06], Classycle [Elm], dependometer [Val], LDM [Lat, SJSJ05], SonarJ [hel], and Sotograph [Sof], evinces the crucial importance of controlling inter-subsystem references in large software projects.

The obvious disadvantage is that the acceptable references need to be specified manually, which is expensive and requires expert knowledge. In contrast, the d-ref leverage of references can be computed automatically.

Large References Most existing design measures quantify attributes of software entities, not attributes of their relationships. If an attribute of subsystem references is measured at all, then it is usually their size (e.g. in the tools Structure101 [Hea] and LDM [SJSJ05, Lat]). The size of a subsystem reference is often defined as the number of corresponding element references, but many further measures can be generated by counting only specific types of references (e.g. function invocation, variable access, type access), by counting referenced or referencing software elements instead of references, and by introducing specific weighting schemes for some types for references (e.g. [BWL99]).

The size of inter-references is not a precise indicator of design flaws. A large (or small) size of an inter-reference primarily indicates that the reference connects two large (or small) subsystems. (The size of a subsystem is not related to design quality, but only to the level of detail at which the subsystem is currently examined.) For all five example software systems, the ten largest inter-references include not a single flawed reference. The precision and recall of the smallest inter-references for detecting flawed references is shown in Figure 11.2b on page 189. A comparison with Figure 11.2a confirms the prediction that a small density of inter-references is a more precise indicator of design flaws than a small size.

Bril and Postma [BP01] actually introduce a measure called directed connectivity that is equivalent to the density of subsystem references, but do not suggest its use as indicator of flawed references (or of other design flaws that could not be detected with the plain size of the references).

Subsystems with Large References Many catalogs of object-oriented design measures include measures for the references of individual classes (e.g. [LH93, CK94, LK94], [HS96, Chapter 6]); see [BDW99] for a survey and classification. Most of these measures basically count the ingoing or outgoing references of the class, in one of the variations mentioned in the previous paragraph, and could be easily generalized from classes to arbitrary subsystems. However, the number of ingoing references, and in particular the number of outgoing references, are strongly correlated with the subsystem size, and subsystem size is generally not related to design quality (as discussed in the previous paragraph).

In a recent paper, Ducasse et al. [DLP05] suggest to divide the number of ingoing references of a subsystem by the number of ingoing and internal references, and similarly for outgoing references. The measures are *not* properly normalized against subsystem size; for graphs with uniform density, the measurement values decrease with increasing subsystem size.

Many refinements of simple reference counting attempt to achieve increased precision at the price of decreased generality, i.e. by addressing only very specific design flaws. For example, software elements with many ingoing *and* many outgoing references are detected as Local Hubs by the tool SA4J [IBM] and as Bottlenecks by the tool Sotograph [Sof]. Such structures induce many *transitive* references, and are therefore discussed in the next chapter. If software elements have more references from or to another subsystem than the one they are actually in, they may be moved to this other subsystem. The corresponding flaw at the level of methods and attributes in classes is called Feature Envy in [Fow00, Chapter 3], listed as Heuristic 2.10 in [Rie96], and detected automatically in [Ciu99, SSL01, Mar04, BDV04, JJ06]. Less related to the present work, because not purely focused on dependencies, are approaches (e.g. [SGM00, Mar04, TK04, SLT06, MGL06]) that combine reference counts with other measures to detect specific design flaws like God Class [Rie96, Heuristic 3.2] or Shotgun Surgery [Fow00, Chapter 3].

Patterns of References Some design flaws are not characterized by a small or large number or density of references, but by particular configurations of references. A prominent example are cycles of references, which induce many transitive references and are therefore discussed in the next chapter. Examples that are specific to object-oriented programs include:

- Classes that inherit both directly and indirectly from another class (Degenerate Inheritance in [BNL05]).
- Classes that reference another class and one of its superclasses (detected as Abstraction Breaking by the tool RevJava [CIB]).
- Violations of the Law of Demeter [LH89], which states that a method should only invoke methods of its own class, of its parameter types, of the attributes types of its class, and of classes of objects it creates. Similarly Message Chains [Fow00, Chapter 3]).

11.4 Summary

- The total d-ref coupling in a software system is the weighted number of references between different subsystems, where each reference is weighted with the size of the referencing and the referenced subsystem. The scaled d-ref coupling is the ratio of the total d-ref coupling to the maximum possible total d-ref coupling.
- The $\frac{\text{total}}{\text{scaled}}$ d-ref coupling equals the $\frac{\text{total}}{\text{scaled}}$ atpair cut of the reference graph.
- The total and scaled d-ref coupling are directly reflected in matrix visualizations.
- The total d-ref coupling equals the additional costs for comprehension and modification caused by the inter-subsystem references, under simplifying assumptions.
- The d-ref leverage of an inter-subsystem reference is inversely proportional to its density.
- The d-ref leverage of references is directly reflected in matrix visualizations.
- The examined software systems contain references with large d-ref leverage, and thus their d-ref coupling can be strongly reduced by removing relatively few and small references.
- In the examined systems, the precision and recall of the d-ref leverage as indicator of flawed inter-subsystem references is fairly high, much higher than for selecting the smallest, the largest, or random references. The precision is limited e.g. for incohesive or facaded subsystems, the recall for small or highly eroded subsystems.
- The addressed problem of detecting flawed inter-subsystem references is widely recognized as crucially important in large software projects.

Chapter 12

T-Ref Coupling and T-Ref Leverage

Layers, observers, abstract factories, acyclic dependencies, independence from implementation details, and many other design patterns and design principles have one thing in common: They eliminate paths of references between software elements or subsystems[1]. This not only explains (partly) their effectiveness in reducing change propagation and easing comprehension, but also suggests a new design principle: Minimize the coupling caused by paths of references. The following sections introduce a measure of this coupling (as indicator of design quality), and a corresponding measure of leverage (as indicator of design flaws). The underlying model of references is the same as in the previous chapter, namely the reference graph.

12.1 T-Ref Coupling

This section introduces a measure of t-ref coupling (with two variants), relates it to the development costs caused by inter-subsystem references, and shows that it subsumes several well-known design principles and design patterns.

12.1.1 Definition

T-ref dependencies differ from d-ref dependencies (defined in Subsection 11.2.1) only in one respect: D-ref dependencies correspond to the edges (except loops) of the reference graph, while t-ref dependencies correspond to the edges (except loops) of the transitive closure of the reference graph. The *transitive closure* G^+ of a directed graph G has the same vertices as G, and contains an edge from vertex v_1 to vertex v_2 if and only if there is a path of edges from v_1 to v_2 in G (i.e. if there are vertices u_1, ..., u_n with $n \geq 0$ such that (v_1, u_1), (u_1, u_2), ..., (u_n, v_2) are edges of G). The transitive closure of the reference graph is denoted as the *transitive reference graph*.

[1] The relation of existing design patterns and design principles to paths of references is detailed in Subsection 12.1.4.

To restate the adapted definitions from Subsection 11.2.1: A subsystem s_1 *t-ref depends* on another subsystem s_2 if there is a path of references from s_1 to s_2, and the *total t-ref coupling* and the *scaled t-ref coupling* in a software system are the total atpair cut and the scaled atpair cut of its transitive reference graph, respectively. These definitions are summarized in Table 12.1.

Software System	Graph
software element	base vertex
size of software element	weight of base vertex
element reference	base edge
size of element reference	weight of base edge
subsystem	cluster
subsystem size	weight of cluster
subsystem reference	cluster edge
size of subsystem reference	weight of cluster edge
t-ref coupling of two subsystems	product of cluster weights
total t-ref coupling	total atpair cut of the trans. closure of the cluster graph
scaled t-ref coupling	scaled atpair cut of the trans. closure of the cluster graph
t-ref cause	set of cluster edges
size of t-ref cause	total weight of set of cluster edges
leverage of t-ref cause	t-sparsity of set of cluster edges

Table 12.1: Graph model of references (upper part, identical to Table 11.1), t-ref coupling (central part), and t-ref leverage (lower part)

12.1.2 Visualization

The transitive reference graph and the reference graph can be shown in a single matrix visualization, because the edges of the former are a superset of the edges of the latter. To distinguish the edges of the reference graph (i.e. the references) from the remaining edges, they are displayed in a different color.

The t-ref coupling is reflected in matrix visualizations of the transitive reference graph like the d-ref coupling is reflected in matrix visualizations of the reference graph: The total t-ref coupling equals the non-white area outside the diagonal elements, and the scaled t-ref coupling equals the fraction of non-white area outside the diagonal elements.

For example, Figure 12.8 on page 220 shows two matrix visualizations of the transitive reference graph for the software system JHotDraw 5.4. The scaled t-ref coupling is 0.914, and thus almost the entire off-diagonal area of the left matrix is non-white. After removing the marked reference, the scaled t-ref coupling decreases to 0.593, and thus not much more than half of the off-diagonal area remains non-white in the right matrix. Here the removal of a single reference between two subsystems eliminates the t-ref dependencies between many further subsystems. On the other hand, the removal of the subsystem reference from samples to framework does not eliminate any t-ref dependency, because there still remain several paths of references from samples to framework (e.g. via util).

12.1.3 Relation to Development Costs

Subsection 11.2.3 showed that under certain simplifying assumptions, the total *d-ref* coupling equals the additional comprehension (or modification) costs that are caused by the inter-subsystem references. In this subsection, the first assumption of Subsection 11.2.3 is replaced with:

1. A comprehension process with the initial subsystem s involves the comprehension of s itself and of all subsystems that can be reached from s via a path of references. In other words, the comprehension of s requires the comprehension of the subsystems that s references, which again requires the comprehension of the subsystems that these subsystems reference, etc.

It is plausible that comprehension processes follow references, because the thorough comprehension of a subsystem requires the comprehension of the identifiers it uses, including those declared in other subsystems. However, it is certainly an exaggeration that comprehension processes follow *all* outgoing references of each involved subsystem; in a sense, this is the opposite extreme of the corresponding assumption in Subsection 11.2.3, that comprehension processes follow *no* outgoing references, except of the initial subsystem.

The above assumption is also applicable to modification processes, with the adaption that modifications propagate mostly backwards along references. The assumption is particularly plausible for reuse, testing, and other activities that generally require compilable subsets of a system, because a subsystem cannot be compiled without the subsystems it references, which again cannot be compiled without the subsystems they reference, etc.

The same calculation as in Subsection 11.2.3, but using the above modified assumption, shows that the reference-caused development costs equal the total t-ref coupling. It is stressed again that this equality is not claimed to hold precisely in practice; the goal is merely to provide evidence for a relation between the total t-ref coupling and design quality.

12.1.4 Relation to Design Principles and (Anti)Patterns

This subsection shows that minimizing the total t-ref coupling subsumes several existing design principles, patterns, and antipatterns. The first paragraph translates design principles into the terminology of this work, and classifies them into four categories: Acyclic References, Layering, Stable References, Abstract References. The following four paragraphs relate these categories to the total t-ref coupling.

Design Principles Martin proposes three principles of package coupling for object-oriented software systems [Mar03, Chapter 20]:

- Acyclic-Dependencies Principle (ADP): Allow no cycles in the package dependency graph.
- Stable-Dependencies Principle (SDP): Depend in the direction of stability.

- Stable-Abstractions Principle (SAP): A package should be as abstract as it is stable.

Martin's dependencies roughly correspond to references, and packages are particular subsystems. Martin suggests that the abstractness of a package can be measured as the fraction of abstract classes among its classes, that stable packages are characterized by many ingoing and few outgoing references, and that instable packages are characterized by few ingoing and many outgoing references. The ADP is addressed in the paragraph on Acyclic References, the SDP in the paragraph on Stable References, and the SAP in the paragraph on Abstract References.

Martin also proposes a two-part principle of module coupling called Dependency-Inversion Principle (DIP) [Mar03, Chapter 11]:

- High-level modules should not depend on low-level modules. Both should depend on abstractions.
- Abstractions should not depend on details. Details should depend on abstractions.

Here the modules can be seen as subsystems. (In Martin's book, they are usually classes, and the abstractions are abstract classes.) This principle is addressed in the paragraph on Abstract References.

Gamma et al. identify two principles of object-oriented design on which many of their design patterns are based [GHJV95, Chapter 1]:

- Program to an interface, not an implementation.
- Favor object composition over class inheritance.

Gamma et al. denote abstract classes as interfaces, the derived concrete classes as implementations, and referencing roughly as "programming to". Therefore, the first principle is addressed in the paragraph on Abstract References. Their main argument for the second principle is that class inheritance exposes a subclass to the details of the parent's implementation, and thus the second principle can be seen as a special case of the first.

Booch [Boo94, Section 6.1] observes that good software architectures tend to have several attributes in common:

- They are constructed in well-defined layers of abstraction, each layer representing a coherent abstraction, provided through a well-defined and controlled interface, and built upon equally well-defined and controlled facilities at lower levels of abstraction.
- There is a clear separation of concerns between the interface and implementation of each layer, making it possible to change the implementation of a layer without violating the assumptions made by its clients.
- The architecture is simple: common behavior is achieved through common abstractions and common mechanisms.

The first two items can be traced back to seminal works of Dijkstra [Dij68] and Parnas [Par72], respectively. Clearly, not all of these attributes are related to t-ref coupling, but the first item is addressed in the paragraphs on Layering and Abstract References.

Acyclic References One of the most widely known antipatterns are cycles of references[2] (e.g. [Par79], [Rie96, Heuristic 2.2], [Lak96, Section 4.6], [Fow01], [RL06, Chapter 3], [SSM06, Section 10.52], [MT07a], Martin's Acyclic-Dependencies Principle). In object-oriented systems, a special case of reference cycles are references from superclasses to their direct or indirect subclasses, because subclasses can reach their superclasses through a path of inheritance references; this is sometimes stated as separate antipattern (e.g. [SSC96], [Rie96, Heuristic 5.2], [SSM06, Section 10.22]).

The minimization of the total t-ref coupling subsumes the avoidance of cyclic references (in the sense defined in Subsection 9.1.1): In a cycle of references, and only in a cycle of references, every subsystem t-ref depends on every other subsystem; thus removing a cycle or splitting it into smaller cycles reduces the total t-ref coupling, all other things being equal (see Figure 12.1a). In other words, removing or splitting cycles is one of several ways to reduce the total t-ref coupling.

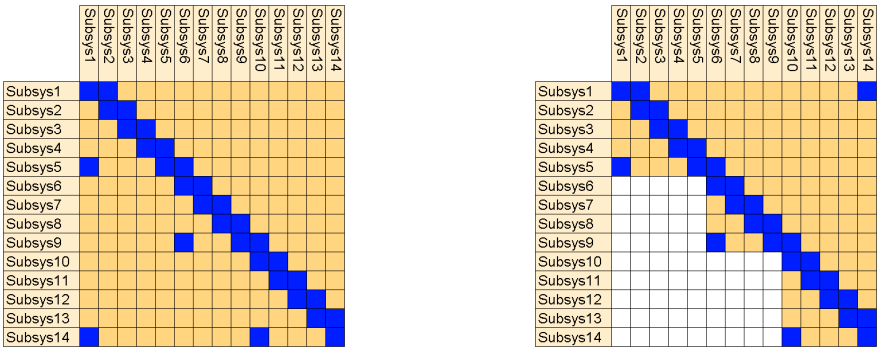
Layering In a layered system, the set of subsystems is partitioned into so-called *layers*, the layers are ordered from highest to lowest, and each layer is only permitted to reference lower layers and itself. Layering is a common architectural pattern ([GS93], [BMR+96, Chapter 2]), and some authors argue that software systems generally should be layered (e.g. [Boo94, Section 6.1], [RL06, Section 3.6]). By its definition, layering prohibits cyclic references between layers, and thus reduces the total t-ref coupling.

A special case of layered architectures are orthogonal architectures, where the system is decomposed not only horizontally into layers, but also vertically into so-called *threads*, such that there are no references between different threads ([RS96], [RL06, Section 3.6.5]). This additional restriction of the references further reduces the total t-ref coupling (see Figure 12.1b).

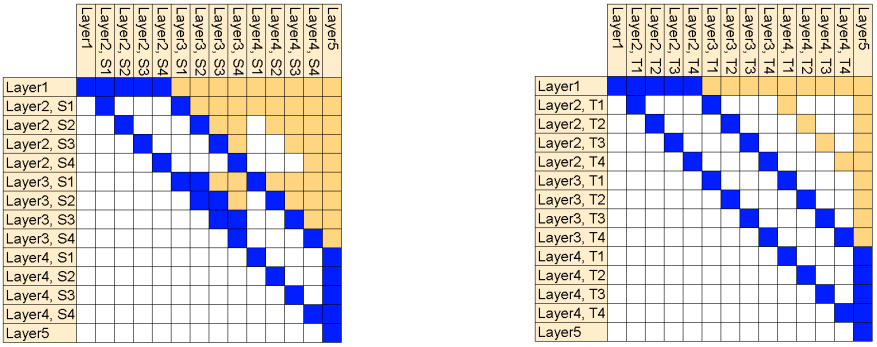
Stable References Martin's Stable-Dependencies Principle discourages references from a stable subsystem, i.e. a subsystem with many ingoing and few outgoing references, to an instable subsystem, i.e. a subsystem with few ingoing and many outgoing references. The minimization of the total t-ref coupling subsumes the avoidance of such instable references: A reference from a stable subsystem s_s to an instable subsystem s_i induces t-ref dependencies from the many subsystems that reference s_s to the many subsystems that are referenced by s_i; thus removing the instable reference or reverting it (to make it stable) potentially reduces the total t-ref coupling significantly (see Figure 12.1c). (However, this reduction is not guaranteed, because there may be further paths of references from the s_s-clients to the s_i-suppliers.)

A similar antipattern are subsystems with many ingoing and many outgoing references, which are detected as Hubs in the tool SA4J [IBM] and as Bottlenecks in the tool Sotograph [Sof].

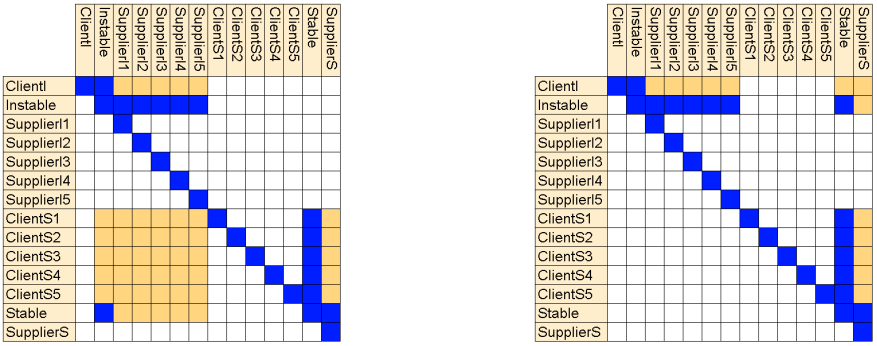
[2]Here the term *cycle* is used for a maximal set of subsystems with a path of references from every subsystem to every other subsystem. Such sets are called strongly connected components in graph theory and tangles in some software analysis tools (e.g. SA4J [IBM] or Structure101 [Hea]). Unless stated otherwise, cycles are assumed to consist of at least two subsystems.



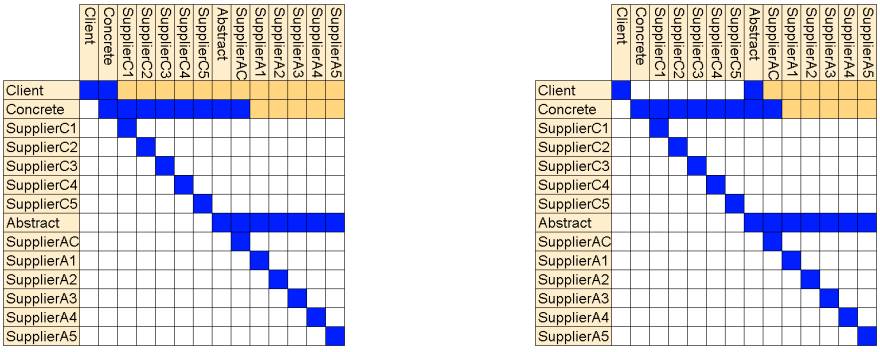
(a) Full cycle vs. split cycle of references



(b) Layered, non-orthogonal system vs. orthogonal system



(c) Instable vs. stable reference



(d) Concrete vs. abstract reference

Figure 12.1: Subsumption of design principles by the total t-ref coupling

Abstract References As detailed in the first paragraph, Booch, Gamma et al., and Martin (among others) suggest to reference abstract subsystems or interfaces rather than concrete subsystems or implementations. Many design patterns of Gamma et al. [GHJV95] are based on this principle, e.g. Abstract Factory, Builder, Bridge, Chain of Responsibility, Decorator, Observer, Strategy. The total t-ref coupling subsumes this principle: Let s_c be a concrete subsystem, let s_a be a corresponding abstract subsystem, and let S_c and S_a be the sets of subsystems on which s_c and s_a t-ref depend, respectively. There is usually a reference from s_c to s_a because implementations reference their interfaces, but no path of references from s_a to s_c (otherwise the references are cyclic, which was discussed above). Thus $S_c \supseteq S_a$ but $S_a \not\supseteq S_c$, and replacing a reference to s_c with a reference to s_a reduces the total t-ref coupling (see Figure 12.1d). (This reduction is not guaranteed, because the removed direct reference to s_c may not be the only path of references to s_c.)

12.1.5 Related Work

For every particular design pattern or antipattern, an indicator of design quality can be obtained simply by counting the occurrences of the (anti)pattern in a software system (e.g. [SSM06]). Of course, such indicators are very specific, because one design principle generally subsumes many (anti)patterns. Even apart from that, counting ignores that different (anti)pattern instances may not be equally beneficial or problematic. For example, the tool Sotograph [Sof] counts the number of reference cycles and the total number of subsystems in reference cycles; however, many small cycles often induce less dependencies than one large cycle (see Figure 12.2).

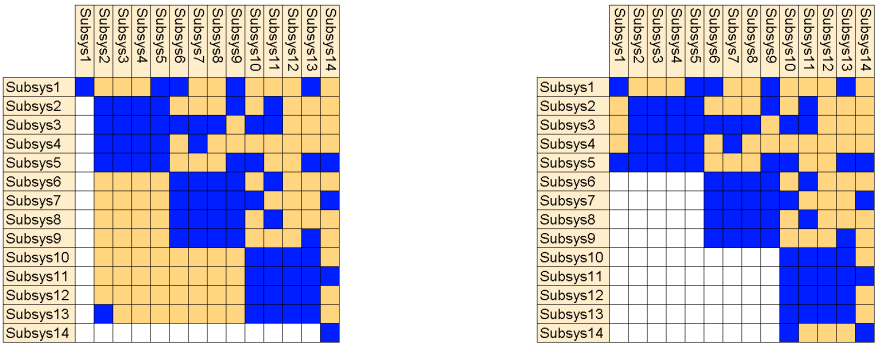


Figure 12.2: Less reference cycles (1 vs. 3) and less subsystems in reference cycles (12 vs. 14) may cause more t-ref coupling

More related to the total t-ref coupling are measures that directly quantify the coupling through paths of references, and indeed, several authors have proposed measures that are equivalent to special cases of the total t-ref coupling.

In Lakos' *cumulative component dependency CCD* [Lak96, Section 4.12], each subsystem consists of one header (`.h`) file and one implementation (`.c`) file in the programming language C++; each subsystem is assumed to have size 1; and, in contrast to the total t-ref coupling, the self-dependencies of the subsystems are also

counted. Like t-ref coupling, CCD is derived as an indicator for the cost of a particular development activity, namely for the costs of linking, which often dominates the cost of regression testing for large systems. Lakos also relates CCD informally to comprehensibility, changeability, and reusability [Lak96, Section 4.13].

Leitch and Stroulia [LS03] propose an indicator of the expected regression-testing costs, which differs from the total t-ref coupling only in three minor respects: It is not based on references, but on a simple notion of data flow and control flow; it includes self-dependencies; and it is normalized with the total size of the system.

The number of classes that can be reached from a class c through a path of references is called *indirect encumbrance* of c by Page-Jones [PJ00, Section 9.2.1] and *class reachability set size* of c by Melton and Tempero [MT07a]. The sum of these numbers over all classes is the special case of the total t-ref coupling where the subsystems are classes and have weight 1.

MacCormack et al.'s *propagation cost* [MRB06] is the special case of the scaled t-ref coupling where all subsystems have weight 1, except that it includes self-dependencies. As the name suggest, the measure is derived from simple assumptions about modifications, namely that changes to a single subsystem propagate backward along references, and all direct and indirect references incur the same cost.

12.2 T-Ref Leverage

This section derives a measure of leverage from the measure of t-ref coupling defined in the previous section, and examines its validity as indicator of design flaws.

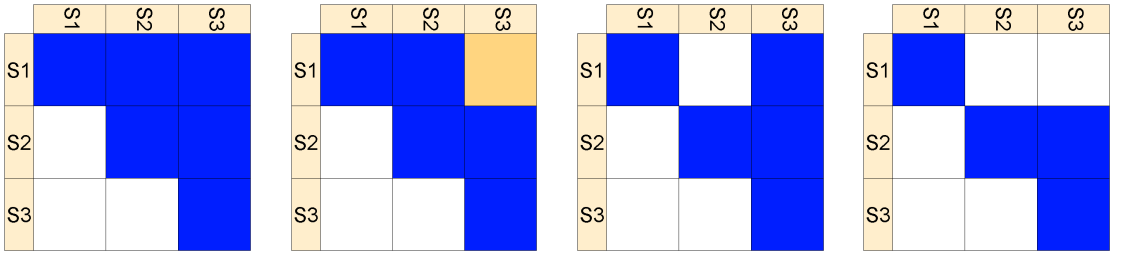
12.2.1 Definition

In this section, *sets* of references between different subsystems (shortly *inter-sets*) are considered as causes of t-ref coupling. Sets of references are preferred over individual references because the amount of t-ref coupling caused by a reference depends on the presence of other references. In the example of Figure 12.3, the removal of the reference (s_1, s_3) from the original system does *not* reduce the total t-ref coupling, but the removal of the same reference does reduce the total t-ref coupling after reference (s_1, s_2) has been removed.

As defined in Section 9.1, a dependency cause has a leverage of $\frac{k_1}{k_2}$ if it causes $k_1\%$ of the total coupling and its size is $k_2\%$ of the total size of all dependency causes. Thus, for the reference graph $G = ((V, \mathrm{w}), (E, \mathrm{f}))$ of a software system, the *t-ref leverage* of an inter-set $C \subset E$ is

$$\frac{\mathrm{P}(G^+) - \mathrm{P}((G-C)^+)}{\mathrm{P}(G^+)} \Big/ \frac{\sum_{(v_1,v_2)\in C} \mathrm{f}(v_1, v_2)}{\sum_{(v_1,v_2)\in E:\, v_1 \neq v_2} \mathrm{f}(v_1, v_2)} \,,$$

where $G-C$ is the result of removing C from G, and P is the total atpair cut. An exponential-time algorithm and several polynomial-time heuristics for computing the inter-sets with the largest t-ref leverage in a given reference graph are described and evaluated in the diploma thesis of Radicke [Rad05].



(a) Original system (b) (s_1, s_3) removed (c) (s_1, s_2) removed (d) (s_1, s_2) and (s_1, s_3) removed

Figure 12.3: Impact of removing references on t-ref coupling

12.2.2 Visualization

In matrix visualizations of a transitive reference graph, an inter-set corresponds to a set of off-diagonal matrix elements. The size of an inter-set is reflected as the total color weight in the corresponding matrix elements, but the caused t-ref coupling is not directly reflected in the visualization. In tools that support the interactive removal of inter-sets, the caused t-ref coupling corresponds to the resulting reduction of the non-white area (as illustrated in Figure 12.3). Even for static visualizations, there are two visual heuristics for identifying inter-sets with large t-ref leverage, which are outlined in the following.

First, sparse references between different subsystems tend to have large t-ref leverage. Such sparse references correspond to off-diagonal matrix elements with small but non-zero color density. By definition, sparse references are small, but connect large subsystems. Because of the latter point, there is a relatively good chance that their removal significantly decreases t-ref coupling. In the left visualization of JHotDraw on page 220, the marked matrix element corresponds to the reference with both the smallest density and the largest t-ref leverage.

Second, backward references tend to have large t-ref leverage, in orderings that minimize the total atedge length[3] of the backward edges. Here a *backward reference* in an ordering is a reference where the referencing subsystem has a greater position than the referenced subsystem. Backward references correspond to matrix elements below the diagonal. The longer the backward references, i.e. the further the matrix elements are placed below the diagonal, the larger tends to be the t-ref leverage. The reason is that deleting all backward references in *any* ordering removes all cycles of references, and cycles of references cause a large amount of t-ref coupling (as discussed in Subsection 12.1.4). The purpose of not using any ordering, but an ordering with a small total atedge length of the backward edges, is to minimize the size of the backward references, particularly of the *long* backward references. In all matrix visualizations of real-world software systems on page 215 to 221, the subsystems are ordered accordingly. In the left visualization of JHotDraw on page 220, the marked matrix element corresponds to both the longest backward reference and the reference with the largest t-ref leverage. ArgoUML on page 215 is another example where the inter-set with the largest t-ref leverage consists of long backward references.

[3]The total atedge length is a quality measure for graph orderings introduced in Chapter 5.

12.2.3 Relation to Design Flaws

This subsection reports the results of an experiment that was performed to test the following two hypotheses (similarly to Subsection 11.3.3 for the d-ref leverage):

- Software systems contain system parts with a t-ref leverage much greater than 1; in other words, small parts of the systems cause a large part of their total t-ref coupling.
- The t-ref leverage indicates design flaws with high precision, both in absolute terms and relative to random selection.

If the t-ref leverage is a precise indicator of design flaws, then the total t-ref coupling is a valid indicator of design quality, in the sense defined in Subsection 9.1.2.

Method The analyzed software systems ArgoUML 0.22, Eclipse 3.1, JDK 1.4.2, JHotDraw 5.4, and JWAM 1.8 are described in Appendix A.3.

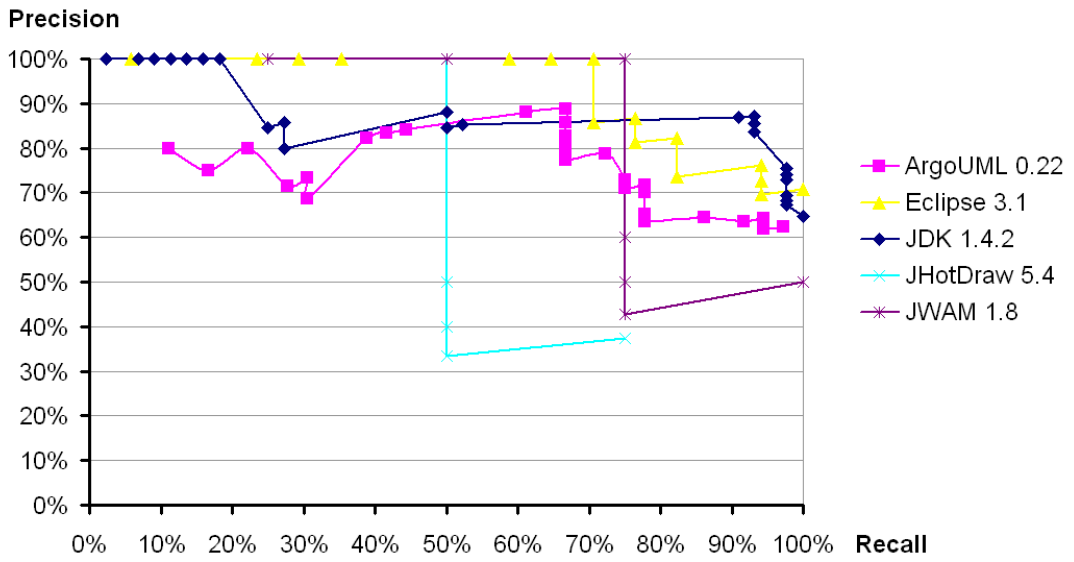
To examine the existence of system parts with large t-ref leverage, the inter-sets with the largest leverage were computed for each system.

To determine the precision of the computed inter-sets, an authoritative classification of the references as flawed or non-flawed is required. This classification was manually derived from the responsibilities of the subsystems; see Appendix A.3 for details.

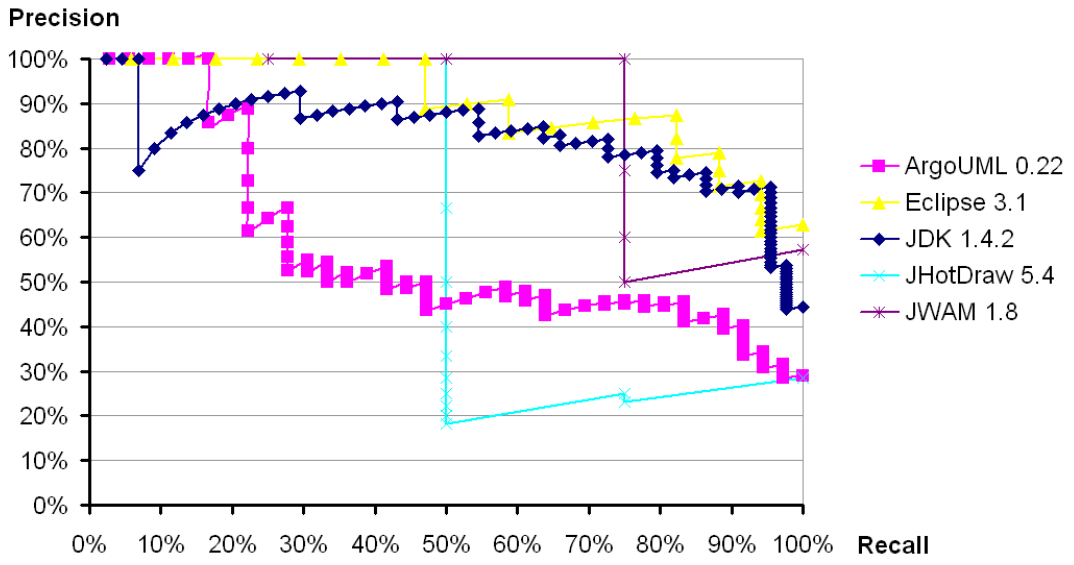
Results The inter-sets with the largest t-ref leverage are listed in Table 12.2 to 12.6 on page 214 to 221. The facing page of each table displays two matrix visualizations of the respective software system, one showing the original system, and one showing the system after removing an inter-set with large t-ref leverage. In the tables, the inter-sets are separated by horizontal lines, and (almost) each inter-set is a superset of all preceding inter-sets; in the few cases where an inter-set does not include a reference from the preceding inter-set, this reference is marked with a minus symbol. In Table 12.2 on page 214, for example, the third and fourth inter-set contain 10 and 14 references, respectively, and the fourth inter-set does *not* contain the reference (uml.ui.ui, uml.reveng).

The tables show that all five software systems contain inter-sets with a t-ref leverage significantly greater than 1. The largest leverage is between 33.3 (for JWAM) and 1 882 (for JDK). Only 0.02% (JDK) to 0.71% (JWAM) of the inter-subsystem references cause 10% of the total t-ref coupling, and only 0.02% (JDK) to 4.04% (JWAM) of the inter-subsystem references cause 25% of the total t-ref coupling.

The precision and recall of the t-ref leverage as indicator of flawed references are summarized in Figure 12.4a. Halve of all flawed references are recalled with at least 84% precision for all systems (and even with optimum precision for three systems), and 95% of all flawed references are recalled with at least 61% precision for the three large systems ArgoUML, Eclipse and JDK. For all systems, the precision of t-ref leverage is much higher than the precision of random selection, which is shown in Figure 11.2c on page 189.



(a) Inter-sets with largest t-ref leverage (Table 12.2 to 12.6)



(b) Inter-sets with largest d-ref leverage (Table 11.2 to 11.6), for comparison

System	Recall	Precision
ArgoUML	28 / 36 = 77.8%	28 / 37 = 75.7%
Eclipse	6 / 17 = 35.3%	6 / 10 = 60.0%
JDK	27 / 44 = 61.4%	27 / 32 = 84.4%
JHotDraw	3 / 4 = 75.0%	3 / 8 = 37.5%
JWAM	1 / 4 = 25.0%	1 / 2 = 50.0%

(c) Refs. to subsystems with smaller stability

Recall	Precision
16 / 36 = 44.4%	16 / 72 = 22.2%
6 / 17 = 35.3%	6 / 10 = 60.0%
21 / 44 = 47.7%	21 / 88 = 23.9%
1 / 4 = 25.0%	1 / 5 = 20.0%
1 / 4 = 25.0%	1 / 4 = 25.0%

(d) Refs. to smaller abstractness

Figure 12.4: Precision and recall of four methods for detecting flawed references

Nr	Inter-Set	Flaw	Size	Caused Coupling	Lev.
1	(uml.ui.ui, language)	yes	10 (0.20%)	165168 (10.4%)	51.5
	(kernel, uml.cognitive)	yes			
	(ui, uml.cognitive)	yes			
	(cognitive, uml.cognitive)	yes			
	(persistence, uml.cognitive)	no			
2	(ui, uml.notation)	yes	16 (0.32%)	218832 (13.7%)	42.7
	(notation, uml.notation)	yes			
	(uml.diagram, uml.notation)	no			
3	(uml.diagram, uml.reveng)	yes	19 (0.38%)	241746 (15.1%)	39.7
	(uml.ui.ui, uml.reveng)	yes			
4	-(uml.ui.ui, uml.reveng)	yes	82 (1.65%)	636003 (39.8%)	24.2
	(kernel, uml.generator)	yes			
	(ui, uml.ui.ui)	yes			
	(cognitive, uml.ui.ui)	yes			
	(uml.diagram, uml.ui.ui)	no			
	(uml.reveng, uml.ui.ui)	no			
5	(uml.ui.ui, uml.reveng)	yes	84 (1.69%)	644841 (40.4%)	23.9
6	(persistence, ocl)	no	85 (1.71%)	648633 (40.6%)	23.8
7	-(persistence, ocl)	no	97 (1.95%)	684657 (42.9%)	22.0
	-(persistence, uml.cognitive)	no			
	(kernel, persistence)	yes			
	(ui, persistence)	yes			
	(uml.ui.ui, persistence)	yes			
8	(uml.cognitive, persistence)	yes	99 (1.99%)	690129 (43.2%)	21.7
9	(uml.cognitive, language)	yes	101 (2.03%)	695409 (43.6%)	21.5
10	(kernel, uml.diagram)	yes	211 (4.24%)	898423 (56.3%)	13.3
	(ui, uml.diagram)	yes			
	(uml.uml, uml.diagram)	yes			
	(uml.ui.ui, uml.diagram)	yes			
	(uml.ui.foundation, uml.diagram)	yes			
	(uml.ui.behavior, uml.diagram)	yes			
11	(kernel, cognitive)	yes	243 (4.88%)	954655 (59.8%)	12.3
	(ui, cognitive)	yes			
12	(uml.diagram, cognitive)	no	253 (5.08%)	971791 (60.9%)	12.0
13	(language, uml.cognitive)	no	258 (5.18%)	979711 (61.4%)	11.8
14	(uml.ui.ui, ocl)	no	260 (5.22%)	982657 (61.6%)	11.8
15	(uml.reveng, cognitive)	no	261 (5.24%)	983953 (61.6%)	11.8
16	(uml.ui.ui, uml.ui.behavior)	yes	310 (6.23%)	1043167 (65.4%)	10.5
	(uml.ui.foundation, uml.ui.behavior)	yes			
17	(kernel, ui)	yes	323 (6.49%)	1058471 (66.3%)	10.2
	(notation, ui)	no			
	(uml.generator, ui)	no			
	(language, cognitive)	no			
18	(persistence, uml.cognitive)	no	327 (6.57%)	1063107 (66.6%)	10.1
19	(uml.ui.ui, uml.ui.model_management)	yes	330 (6.63%)	1066495 (66.8%)	10.1
20	(uml.uml, util)	no	331 (6.65%)	1067615 (66.9%)	10.1
21	(ui, util)	no	341 (6.85%)	1078319 (67.6%)	9.86
	(cognitive, util)	no			
	(uml.diagram, util)	no			
22	(uml.ui.ui, uml.util)	no	343 (6.89%)	1080283 (67.7%)	9.82
23	-(language, uml.cognitive)	no	426 (8.56%)	1156468 (72.5%)	8.47
	-(language, cognitive)	no			
	(kernel, notation)	yes			
	(ui, notation)	yes			
	(cognitive, ui)	no			
	(uml.cognitive, ui)	no			
	(uml.cognitive, uml.diagram)	no			
	(uml.ui.ui, uml.generator)	yes			
24	(kernel, uml.uml)	yes	440 (8.84%)	1169221 (73.2%)	8.29
	(ui, uml.uml)	yes			
	(uml.ui.ui, uml.uml)	no			
	(uml.diagram, uml.uml)	no			
25	(cognitive, uml.uml)	yes	442 (8.88%)	1170805 (73.3%)	8.26
26	(persistence, uml.diagram)	no	466 (9.36%)	1187601 (74.4%)	7.95
	(persistence, ui)	no			
27	(uml.ui.ui, uml.ui.foundation)	yes	493 (9.90%)	1205311 (75.5%)	7.62
...	...	...	...	...	...

Table 12.2: Inter-sets in ArgoUML 0.22, ordered by t-ref leverage. The listed sets include all flawed references except (pattern, uml.cognitive).

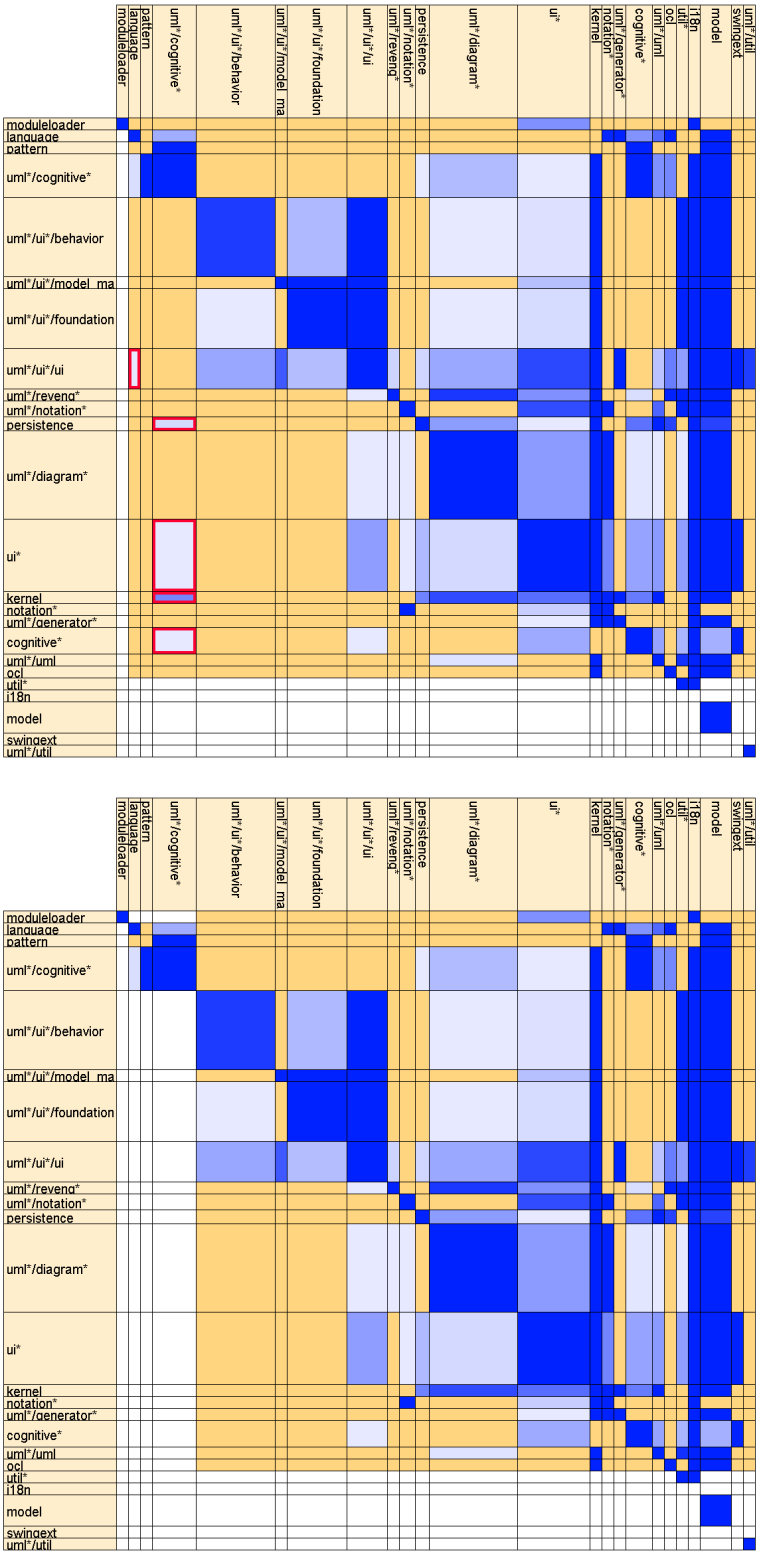


Figure 12.5: T-ref dependencies in ArgoUML 0.22 before and after removing 0.20% of all inter-subsystem references (number 1 in Table 12.2). The t-ref coupling decreases by 10.4% from 1 596 328 (total) or 0.916 (scaled) to 1 431 160 or 0.821.

Nr	Inter-Set	Flaw	Size	Caused Coupling	Lev.
1	(JDT UI, Team)	yes	1 (0.002%)	817920 (1.24%)	769
2	(Runtime, Workbench)	yes	27 (0.04%)	8308796 (12.6%)	289
	(Workbench, Update)	yes			
	(Runtime, Update)	yes			
3	(PDE Build, Ant)	yes	29 (0.05%)	8661456 (13.1%)	281
4	(Runtime, SWT)	yes	33 (0.05%)	9125612 (13.8%)	260
5	(Workbench, Help)	yes	60 (0.10%)	11604460 (17.5%)	182
	(Debug, Help)	yes			
	(JDT UI, Help)	yes			
	(PDE UI, Help)	yes			
6	(PDE UI, Ant)	yes	67 (0.11%)	11968679 (18.1%)	168
7	(PDE UI, Team)	yes	74 (0.12%)	12214695 (18.5%)	155
8	(PDE UI, JDT Debug)	no	156 (0.25%)	14966038 (22.6%)	90.2
	(PDE UI, JDT UI)	no			
9	(Workbench, Text)	yes	210 (0.34%)	16609306 (25.1%)	74.3
10	(JDT Core, Text)	no	330 (0.53%)	20043176 (30.3%)	57.1
11	(JDT UI, JDT Debug)	yes	388 (0.62%)	21641009 (32.7%)	52.4
12	(PDE UI, PDE Build)	no	405 (0.65%)	22058083 (33.3%)	51.2
	(PDE UI, Update)	no			
13	(Ant, JDT Debug)	yes	474 (0.76%)	23194325 (35.0%)	46.0
	(Ant, JDT UI)	yes			
14	(JDT UI, Debug)	no	546 (0.88%)	24370269 (36.8%)	41.9
15	(Update, Workbench)	no	635 (1.02%)	25322031 (38.3%)	37.5
16	(Ant, JDT Core)	yes	677 (1.09%)	25718086 (38.9%)	35.7
17	-(Ant, JDT Core)	yes	797 (1.28%)	26813565 (40.5%)	31.6
	-(Ant, JDT Debug)	yes			
	(JDT Debug, JDT UI)	no			
18	(Compare, Text)	no	865 (1.39%)	27337689 (41.3%)	29.7
	(CVS, Text)	no			
19	(Ant, JDT Debug)	yes	972 (1.56%)	28050967 (42.4%)	27.1
	(Ant, JDT Core)	yes			
20	(PDE UI, Debug)	no	1083 (1.74%)	28642943 (43.3%)	24.9
21	(Debug, Text)	no	1163 (1.87%)	29032871 (43.9%)	23.5
22	(Text, Workbench)	no	1429 (2.30%)	30239369 (45.7%)	19.9
23	(PDE UI, JDT Core)	no	1687 (2.71%)	31243614 (47.2%)	17.4
24	(PDE UI, Search)	no	1721 (2.77%)	31373349 (47.4%)	17.1
25	(JDT UI, Search)	no	1801 (2.89%)	31631064 (47.8%)	16.5
26	(JFace, Runtime)	no	1865 (3.00%)	31807800 (48.1%)	16.0
27	(Help, Workbench)	no	2083 (3.35%)	32369208 (48.9%)	14.6
28	(JDT Debug, Text)	no	2302 (3.70%)	32899029 (49.7%)	13.4
29	(Compare, Workbench)	no	2397 (3.85%)	33122031 (50.0%)	13.0
30	(PDE UI, Text)	no	2669 (4.29%)	33730344 (51.0%)	11.9
...	...	...	...	...	...

Table 12.3: Inter-sets in Eclipse 3.1, ordered by t-ref leverage. The listed sets include all flawed references.

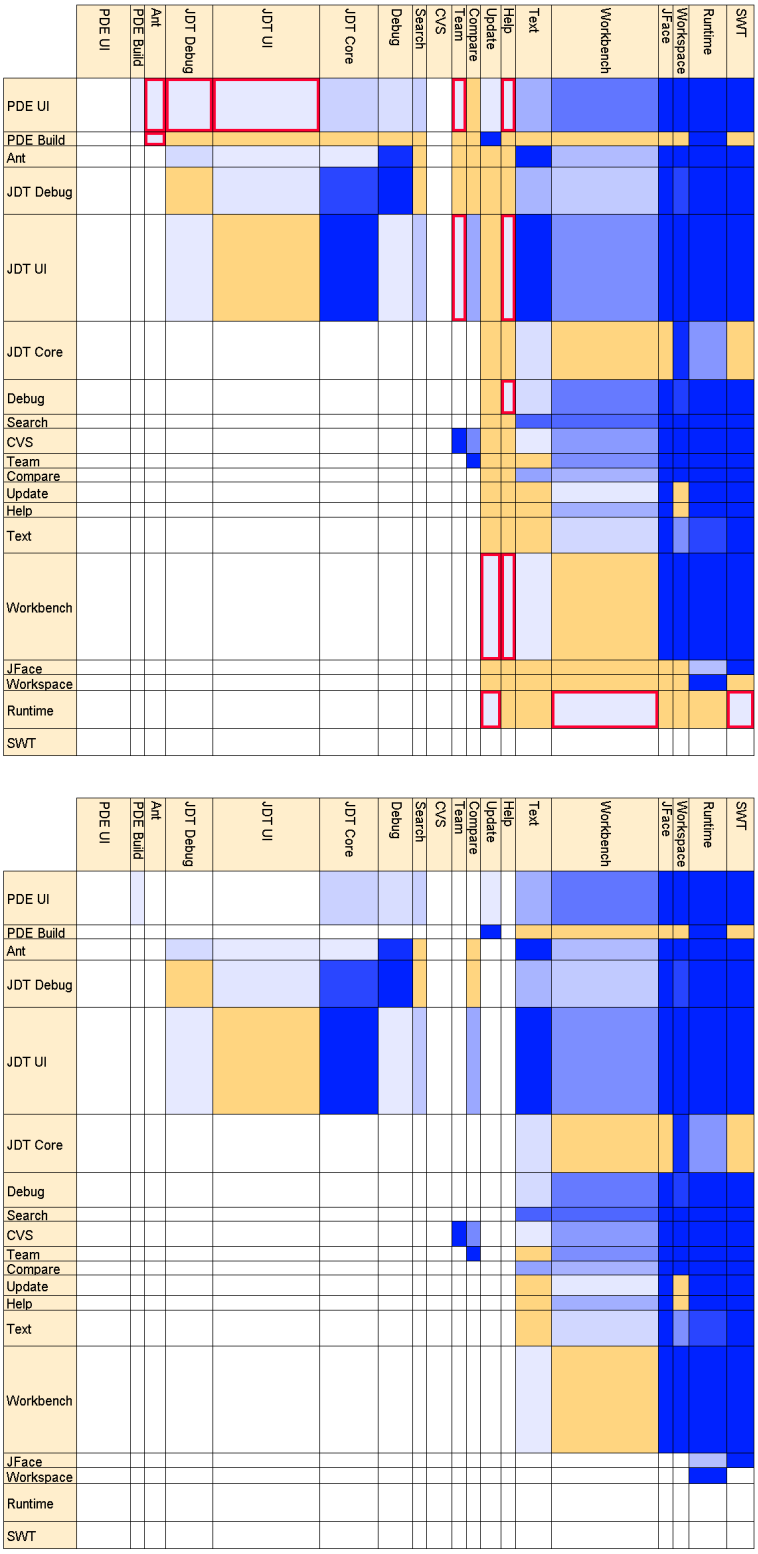


Figure 12.6: T-ref dependencies in Eclipse 3.1 before (top) and after removing 0.25% of all inter-subsystem references (number 1 to 8 in Table 12.3). The t-ref coupling decreases by 22.6% from 66.2 million (total) or 0.537 (scaled) to 51.2 million or 0.415.

Nr	Inter-Set	Flaw	Size	Caused Coupling	Lev.
1	(java.text, java.awt)	yes	2 (0.02%)	1223785 (33.3%)	1882
2	(java.awt, javax.swing)	yes	5 (0.04%)	1566505 (42.6%)	963
	(javax.accessibility, javax.swing)	yes			
3	(java.awt, java.rmi)	yes	6 (0.05%)	1640293 (44.6%)	841
4	(java.awt, javax.print)	yes	11 (0.10%)	1761805 (47.9%)	493
5	(java.util, javax.xml)	yes	14 (0.12%)	1825735 (49.6%)	401
6	(javax.print, java.awt)	yes	17 (0.15%)	1872217 (50.9%)	339
7	(java.security, javax.security)	yes	23 (0.20%)	1950805 (53.0%)	261
8	(java.lang, java.nio)	yes	52 (0.46%)	2211109 (60.1%)	131
	(java.io, java.nio)	yes			
	(java.util, java.nio)	yes			
	(java.awt, java.nio)	no			
	(java.net, java.nio)	no			
9	(java.util, java.text)	yes	59 (0.52%)	2260881 (61.4%)	118
10	(javax.swing, javax.sound)	no	65 (0.57%)	2291013 (62.3%)	108
11	(java.lang, java.net)	yes	88 (0.78%)	2365049 (64.3%)	82.7
	(java.io, java.net)	yes			
	(java.util, java.net)	yes			
	(java.text, java.net)	yes			
	(java.nio, java.net)	yes			
	(java.security, java.net)	yes			
	(java.awt, java.applet)	yes			
	(java.awt, java.net)	yes			
	(javax.imageio, java.net)	yes			
	(javax.naming, java.net)	yes			
12	(javax.imageio, java.nio)	no	91 (0.80%)	2372573 (64.5%)	80.2
13	(javax.security, java.net)	yes	92 (0.81%)	2374793 (64.5%)	79.4
14	(java.lang, java.security)	yes	311 (2.75%)	2699459 (73.4%)	26.7
	(java.io, java.security)	yes			
	(java.util, java.security)	yes			
	(java.text, java.security)	yes			
	(java.nio, java.security)	yes			
	(java.sql, java.security)	yes			
	(java.awt, java.security)	yes			
	(javax.print, java.security)	yes			
	(javax.sound, java.security)	yes			
	(javax.swing, java.security)	yes			
	(javax.imageio, java.security)	yes			
	(javax.naming, java.security)	yes			
	(java.applet, java.net)	yes			
	(java.sql, java.net)	yes			
	(javax.print, java.net)	yes			
	(javax.sound, java.net)	yes			
	(javax.swing, java.net)	yes			
	(java.text, java.math)	no			
	(java.awt, java.math)	no			
15	(javax.accessibility, java.awt)	yes	320 (2.83%)	2708659 (73.6%)	26.0
16	(java.awt, java.text)	no	343 (3.03%)	2727749 (74.1%)	24.5
17	(javax.imageio, java.awt)	no	375 (3.31%)	2749238 (74.7%)	22.6
18	(java.lang, java.util)	yes	490 (4.33%)	2823296 (76.7%)	17.7
	(java.io, java.util)	yes			
	(java.nio, java.util)	no			
	(javax.sound, java.util)	no			
	(javax.xml, java.util)	no			
	(java.rmi, java.util)	no			
	(java.rmi, java.net)	no			
	(java.rmi, java.security)	no			
19	(java.math, java.util)	no	491 (4.34%)	2823926 (76.7%)	17.7
20	(javax.security, java.text)	no	494 (4.36%)	2825628 (76.8%)	17.6
21	-(java.text, java.math)	no	547 (4.83%)	2851066 (77.5%)	16.0
	(javax.swing, java.text)	no			
22	(java.applet, java.awt)	no	560 (4.95%)	2855884 (77.6%)	15.7
	(java.applet, java.util)	no			
	(javax.accessibility, java.util)	no			
23	(javax.print, java.util)	no	602 (5.32%)	2871256 (78.0%)	14.7
24	(javax.naming, java.util)	no	643 (5.68%)	2883352 (78.3%)	13.8
25	(java.lang, java.io)	yes	745 (6.58%)	2912232 (79.1%)	12.0
	(java.nio, java.io)	no			
	(javax.accessibility, java.io)	no			
	(javax.naming, java.io)	no			

Table 12.4: Inter-sets in JDK 1.4.2, including all flawed references

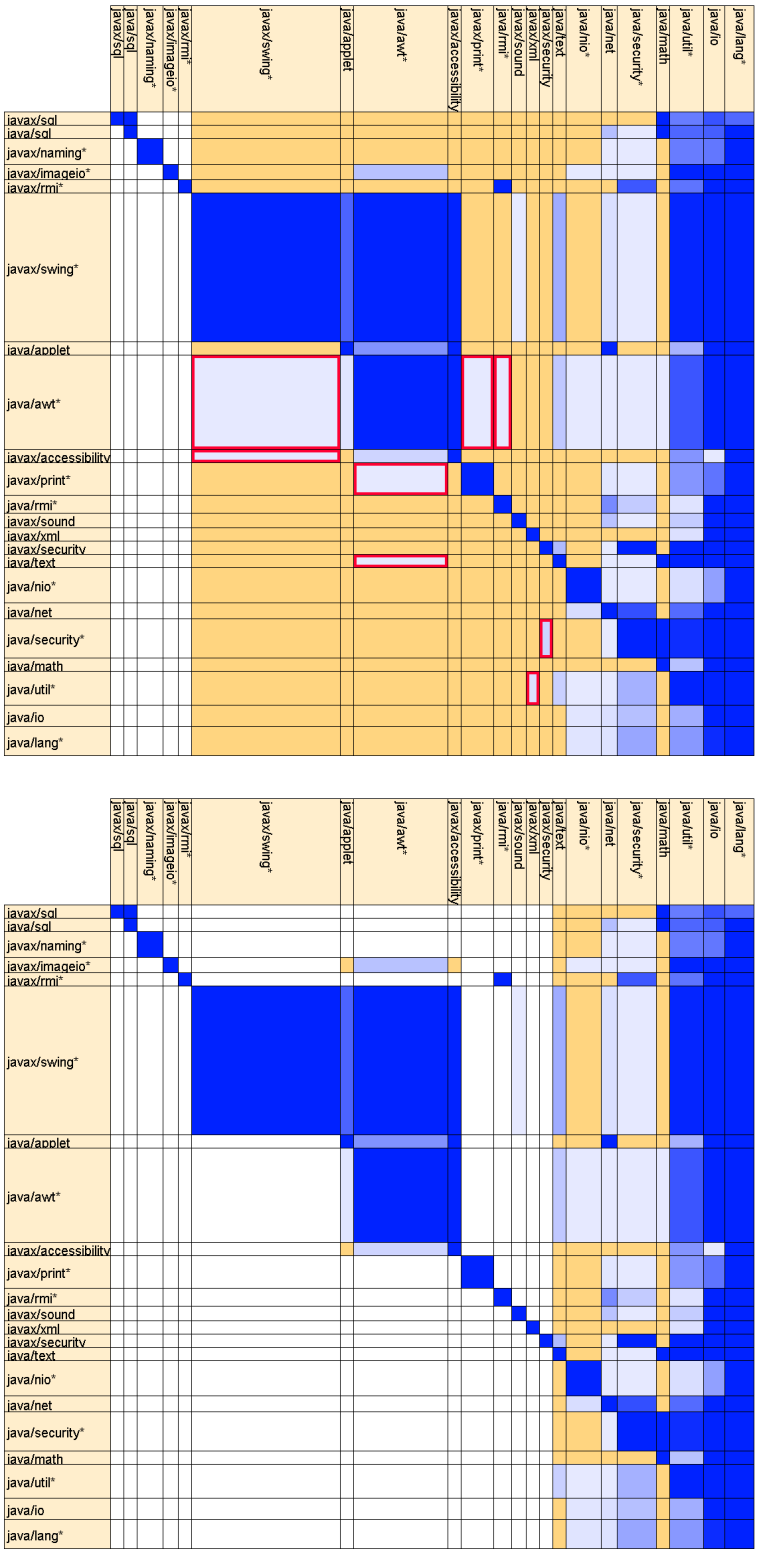


Figure 12.7: T-ref dependencies in JDK 1.4.2 before and after removing 0.20% of all inter-subsystem references (number 1 to 7 in Table 12.4). The t-ref coupling decreases by 53.0% from 3 680 346 (total) or 0.894 (scaled) to 1 729 541 or 0.420.

Nr	Inter-Set	Flaw	Size	Caused Coupl.	Lev.
1	(standard, contrib)	yes	4 (0.36%)	22772 (35.1%)	98.4
2	(util, standard)	yes	12 (1.07%)	28174 (43.4%)	40.6
3	(application, figures)	no	27 (2.40%)	31594 (48.6%)	20.2
	(contrib, figures)	no			
4	(framework, util)	no	33 (2.94%)	32842 (50.5%)	17.2
5	-(application, figures)	no	92 (8.19%)	39762 (61.2%)	7.47
	(contrib, application)	no			
	(contrib, standard)	no			
6	(application, contrib)	yes	122 (10.9%)	41830 (64.4%)	5.93
	(samples, contrib)	no			
...	...	...	...	...	...

Table 12.5: Inter-sets in JHotDraw 5.4, ordered by t-ref leverage. The listed sets include all flawed references except (util, framework).

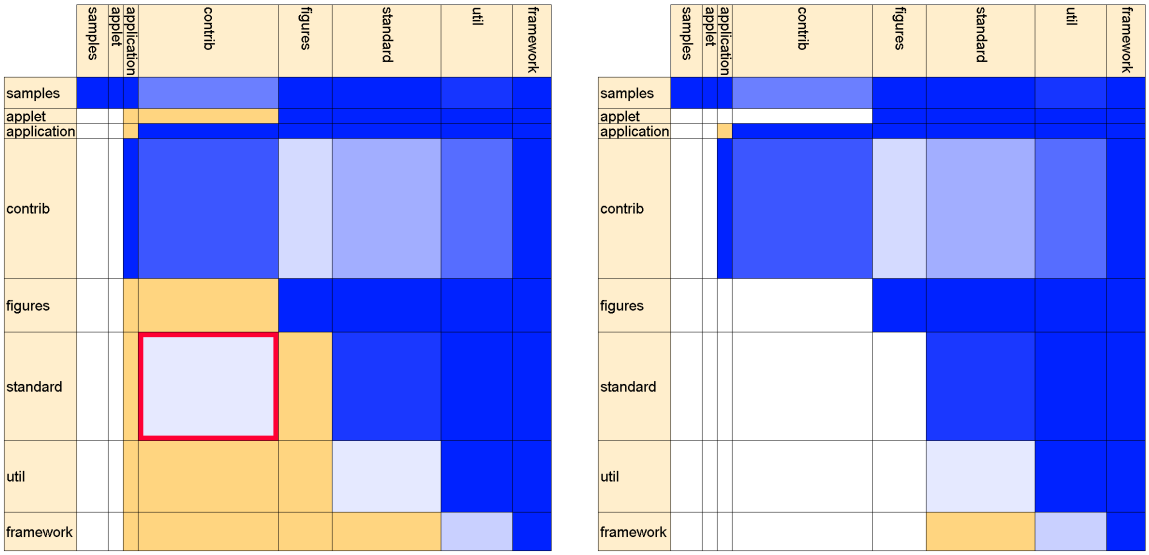


Figure 12.8: T-ref dependencies in JHotDraw 5.4 before (left) and after (right) removing 0.36% of all inter-subsystem references (number 1 in Table 12.5). The t-ref coupling decreases by 35.1% from 64 979 (total) or 0.914 (scaled) to 42 207 or 0.593.

Nr	Inter-Set	Flaw	Size	Caused Coupl.	Lev.
1	(jwamx.handling, jwambeta.handling)	yes	2 (0.10%)	6208 (3.37%)	33.3
2	(jwamdev, jwambeta.handling)	yes	5 (0.25%)	14970 (8.13%)	32.2
3	(jwamx.technology, jwam.handling)	yes	14 (0.71%)	29335 (15.9%)	22.5
4	(jwamx.technology, jwamx.lang)	no	21 (1.06%)	32562 (17.7%)	16.7
	(jwamx.handling, jwamx.lang)	no			
5	(jwamdev, jwam.handling)	no	27 (1.37%)	34772 (18.9%)	13.8
6	(jwambeta.handling, jwamx.handling)	no	49 (2.48%)	40980 (22.3%)	8.98
7	(jwamexample, jwambeta.handling)	yes	80 (4.04%)	49364 (26.8%)	6.63
8	(jwam.technology, jwam.lang)	no	87 (4.40%)	50995 (27.7%)	6.29
9	(jwambeta.handling, jwamx.technology)	no	136 (6.88%)	61771 (33.5%)	4.88
...	...	...	...	...	...

Table 12.6: Inter-sets in JWAM 1.8, ordered by t-ref leverage. The listed sets include all flawed references.

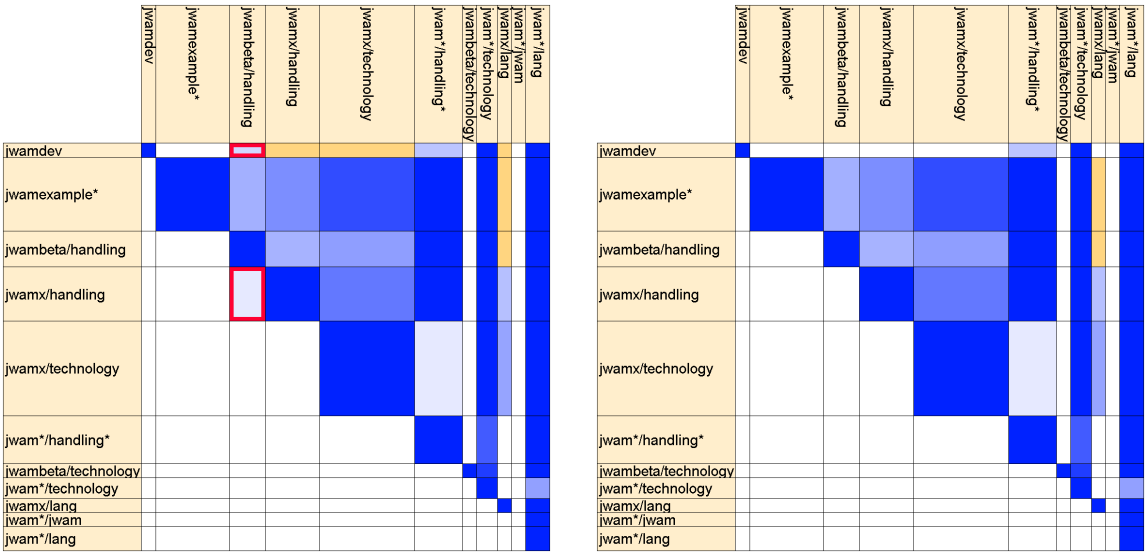


Figure 12.9: T-ref dependencies in JWAM 1.8 before (left) and after (right) removing 0.25% of all inter-subsystem references (number 1 and 2 in Table 12.6). The t-ref coupling decreases by 8.13% from 184 174 (total) or 0.499 (scaled) to 169 204 or 0.458.

Compared to the d-ref leverage (see Figure 12.4b, which equals Figure 11.2a), the precision of the t-ref leverage is much better for ArgoUML, and similar to slightly better for the other four systems. The reasons for the misclassified references are similar to those identified for the d-ref leverage in Subsection 11.3.3.

Discussion The results clearly support the two hypotheses, namely the existence of system parts with large t-ref leverage, and the validity of the t-ref leverage as indicator of design flaws. Altogether, the precision of the t-ref leverage for the five example systems are about as large as one could reasonably hope, given that the definite classification of references as flawed is a highly knowledge-intensive and slightly fuzzy task.

The limitations of the generalizability are the same as in Subsection 11.3.3. As the only difference, the granularity of the subsystem decomposition may be less critical for the t-ref leverage than for the d-ref leverage, because the t-ref leverage is not exclusively based on local properties of individual references.

12.2.4 Related Work

The final four paragraphs of Subsection 12.1.4 related existing design principles and design antipatterns to the minimization of the t-ref coupling. The following four paragraphs discuss existing applications of these principles and antipatterns to the detection of design flaws.

Acyclic References Cycles of references are detected by many academic and commercial tools[4]. The detection of *causes* of cycles, i.e. of small system parts whose modification eliminates cycles, is less widely supported. But such support is essential, as it is not unusual that a single cycle consists of hundreds of software elements. For example, 19 of the 24 subsystems of ArgoUML form one huge cycle (see Figure 12.5).

The tool Classycle [Elm] computes not only cycles of subsystems, but also the best fragmenter of each cycle. The *best fragmenter* is the subsystem in the cycle that minimizes the maximum fragment size. The maximum fragment size of a subsystem in a cycle is the size of the largest remaining cycle after the subsystem has been removed. (Of course, it is difficult to remove a subsystem without changing the behavior of the system, but a similar reduction of cycles can often be achieved by an appropriate split of the subsystem, see e.g. [RL06, Section 4.3]). There are three differences between the best fragmenters of Classycle and inter-sets with large t-ref leverage. First, best fragmenters are subsystems, while inter-sets contain references between subsystems. Second, best fragmenters are *individual* subsystems, while inter-sets are *sets* of references; the reason for using sets was explained in Subsection 12.2.1, and applies for subsystems similarly as for references. Third, best fragmenters only address cycles, while t-ref leverage addresses t-ref coupling in general (including, but not limited to, cycles).

[4]As before, the term cycle denotes a maximal set of subsystems where every subsystem is connected to every other subsystem through a path of references.

A *feedback set* of the reference graph is a set of references whose removal makes the reference graph acyclic. Algorithms for computing feedback sets are discussed by Briand et al. [BLW03], and are implemented in the tools JooJ [MT07b], OptimalAdvisor [Hau02, Com], and Structure101 [Hea]. Computing the feedback set with minimum total size is generally intractable because the corresponding decision problem is NP-complete even for graphs with unit edge weights [GJ79, Problem GT8]. There are two major differences between feedback sets and inter-sets with large t-ref leverage. First, the inter-sets with the largest leverage are often much smaller than feedback sets, but already drastically reduce the cycles. In the JDK, for example, removing references of total size 23 already reduces the cycles drastically (see Table 12.4 and Figure 12.7), while the minimum feedback set contains much larger references, e.g. (java.lang, java.io) with size 48, (java.lang, java.util) with size 37, and (java.lang, java.security) with size 37. Second, feedback sets address only cycles, and not other causes of t-ref coupling.

An improvement of feedback sets has been introduced in Subsection 12.2.2: orderings of the subsystems that minimize the total atedge length of the backward references. Such orderings not only show a set of references (namely, the backward references) whose removal eliminates *all* cycles, but also much smaller sets of references (namely, the longest backward references) whose removal is likely to eliminate *most* cycles. (Similar orderings are provided by the tool LDM [SJSJ05, Lat], but the documentation does not specify how the crucial order of the subsystems within each cycle is computed, or whether this order is arbitrary.) An initial empirical evaluation was performed in the diploma thesis of Huras [Hur06], but is not discussed or extended here because it only addresses cycles, and not t-ref coupling in general.

Layering Violations of layering and orthogonal architectures can be easily detected if the assignment of software elements to layers and threads is given. This is a special case of the manual specification of acceptable references discussed in Section 11.3.4, and has the same disadvantage: It requires knowledge and effort.

Stable References Martin's Stable-Dependencies Principle [Mar03, Chapter 20] discourages references from stable to instable subsystems. Martin considers stability as the amount of work required to make a change (as opposed to the inverse frequency of change), and argues that heavily referenced subsystems are particularly difficult to change. He thus defines the *instability* of a subsystem as $\frac{n_e}{n_a+n_e}$, where n_e is the number of software elements inside the subsystem that reference software elements outside the subsystem, and n_a is the number of software elements outside the subsystem that reference software elements inside the subsystem. A reference violates the Stable-Dependencies Principle if the instability of the referencing subsystem is smaller than the instability of the referenced subsystem.

Figure 12.4c on page 213 shows how precisely these violations indicate flawed references in the example systems. Compared to the t-ref leverage (see Figure 12.4a), the precision for equal recalls is slightly better only for ArgoUML (75.7% vs. 71.8%), and much worse for Eclipse (60% vs. 100%) and JWAM (50% vs. 100%).

Besides Martin's measure, there are other heuristics for estimating the instability of subsystems. For example, subsystems that implement a user interface (UI) are often considered as particularly instable, and thus references from non-UI subsystems to UI subsystems are considered as undesirable (e.g. [Rie96, Heuristic 3.5]). UI subsystems can be automatically recognized from the use of specific UI libraries.

Similar to violations of Martin's Stable-Dependencies Principle are subsystems with many ingoing and many outgoing references, which are detected as Hubs by the tool SA4J [IBM] and as Bottlenecks by the tool Sotograph [Sof]. Hubs are often not design flaws, but simply large subsystems (which tend to have more references than small subsystems), or subsystems in medium layers of layered systems (which are referenced by higher layers and reference lower layers).

Abstract References Many programming languages provide specific constructs for distinguishing abstract subsystems or interfaces from concrete subsystems or implementations, which make the identification of references to implementations trivial. For example, object-oriented languages often distinguish abstract classes from concrete classes; however, by far not all references to concrete classes are design flaws.

For subsystems with more than one class, a combination of Martin's Stable-Dependencies Principle and Stable-Abstractions Principle yields that subsystems should not reference less abstract subsystems [Mar03, Chapter 20]. Martin suggests to measure the *abstractness* of a subsystem simply as the fraction of abstract classes among all classes in the subsystem. Figure 12.4d on page 213 shows that references from more abstract to less abstract subsystems are rarely flawed references, at least for the five example systems.

12.3 Summary

- T-ref dependencies are the irreflexive transitive closure of d-ref dependencies. Otherwise, t-ref coupling is analogous to d-ref coupling, and is similarly easy to measure and visualize.
- The minimization of t-ref coupling subsumes several existing design principles and design patterns, e.g. of Martin and Gamma et al. In particular, it subsumes the minimization of cycles of references, which is widely recognized as crucial.
- Large t-ref leverage is not directly reflected in matrix visualizations, but matrix elements with small color density and matrix elements below the diagonal (assuming an ordering with short backward edges) are promising candidates.
- The examined software systems contain sets of references with large t-ref leverage, and thus their t-ref coupling can be drastically reduced by removing relatively few and small references.
- In the examined software systems, the precision and recall of t-ref leverage as indicator of flawed references is high, slightly higher than for d-ref leverage and for references to larger instability (as defined by Martin), and much higher than for references to smaller abstractness (as defined by Martin).

Chapter 13

Conclusion and Discussion

The goal of Part II was to introduce simple coupling measures that are general (though not exhaustive) indicators of design quality. Three measures, each with two or three variants, were proposed: the total cochange coupling, the total d-ref coupling, and the total t-ref coupling.

13.1 Unification

The generality of indicators for design quality was demonstrated by showing that they subsume several existing design principles, design rules, and design (anti)patterns. A measure is said to *subsume* a design rule or design (anti)pattern if improving the conformance to the rule, introducing the design pattern, or removing the antipattern significantly reduces the measurement value (all other things being equal).

The minimization of the total cochange coupling formalizes the principle of locality of change, and thus subsumes many design principles (e.g. information hiding), design rules, design patterns (e.g. of Gamma et al.), and design antipatterns (e.g. of Beck and Fowler) that address locality of change (see Subsection 10.2.4).

The minimization of the total t-ref coupling subsumes Martin's principles of package coupling, and several design patterns and antipatterns (see Subsection 12.1.4).

More precisely, the total cochange coupling and the total t-ref coupling subsume only specific interpretations of the respective principles, rules, and (anti)patterns; any formalization of originally informal concepts like design principles can only capture one of many possible interpretations. However, the proposed measures are not only particularly simple, but – as discussed in the next section – also quite effective and not entirely arbitrary.

The formalization of design principles and design rules as measures, even without any generalization, is already useful, because it allows to quantify, visualize, and optimize the conformance to the principles and rules. The unification of several design rules and design (anti)patterns into a single measure has additional benefits:

- For the evaluation of designs, a single general measure combines several specific measures derived from individual rules or (anti)patterns. Of course, the specific measures could also be combined arithmetically, but this generally leads to non-interpretable results. For example, instead of measuring the total t-ref coupling, one could add the number of instable references and the number of cycles of references, but the resulting number is meaningless.
- For the improvement of designs, the optimization of a single general measure combines the detection and prioritization of various rule violations and antipattern instances. For example, the incremental optimization of the total t-ref coupling not only combines the detection of cyclic references, instable references, and concrete references; it also prioritizes the detected problems, which would be otherwise difficult due to their diversity.
- The general measures subsume design rules and design (anti)patterns that have not yet been proposed in the literature. For example, violations of Martin's principles of package coupling – namely cyclic references, instable references, and concrete references – are not the only structures that strongly increase the total t-ref coupling; and Gamma et al.'s design patterns are not the only structures that can significantly reduce the total cochange coupling.

However, the subsumed rules and (anti)patterns are clearly not obsolete:

- Specific rules and patterns may have additional benefits, beyond reducing the total cochange coupling or the total t-ref coupling. For example, the relative stability and comprehensibility of well-designed interfaces (compared to implementations) is not only based on their smaller number of outgoing reference paths; thus the full benefit of referencing interfaces instead of implementations may not be adequately reflected by the resulting reduction of the total t-ref coupling.
- The application of specific rules or (anti)patterns may have practical advantages. For example, computing the total cochange coupling requires data about logical changes, and optimizing the total t-ref coupling can be computationally expensive.
- The removal of specific antipatterns, or the introduction of specific patterns, may require specific restructuring strategies.

13.2 Validation

Indicators of design quality were required to be valid in the following sense: If a small design change (excluding changes of the subsystem decomposition) significantly decreases the indicator, then it improves design quality. The opposite requirement – if design quality improves, then the indicator decreases – would be very difficult to satisfy, because design quality has more aspects than a simple indicator can capture.

With this notion of validity, a valid indicator of design quality is suitable for comparing variations of the same design, in order to find a variation that improves design quality. It is not necessarily suitable for comparing different subsystem decompositions or entirely different designs. The total cochange coupling, the total d-ref coupling, and the total t-ref coupling are indeed inappropriate for comparing

substantially different subsystem decompositions, because they are biased towards decompositions with specific granularities (see Subsections 3.1.2 and 7.1.2). The scaled cochange coupling is not biased towards any decomposition, and thus potentially allows the comparison of different decompositions, but this was not evaluated.

Three types of evidence were presented for the validity of the total cochange coupling, the total d-ref coupling, and the total t-ref coupling as indicators of design quality.

Decreasing the total cochange coupling and the total t-ref coupling conforms to several widely accepted design principles, design rules, and design (anti)patterns (see Subsections 10.2.4 and 12.1.4).

This was already discussed in the previous section.

The total cochange coupling, the total d-ref coupling, and the total t-ref coupling quantify the additional costs for modification, comprehension, reuse, or testing that are caused by the respective type of dependency, under simplifying assumptions (see Subsections 10.2.3, 11.2.3, and 12.1.3).

The simplifying assumptions are considered to be self-evident at least as rough approximations of reality; they have not been verified in this work.

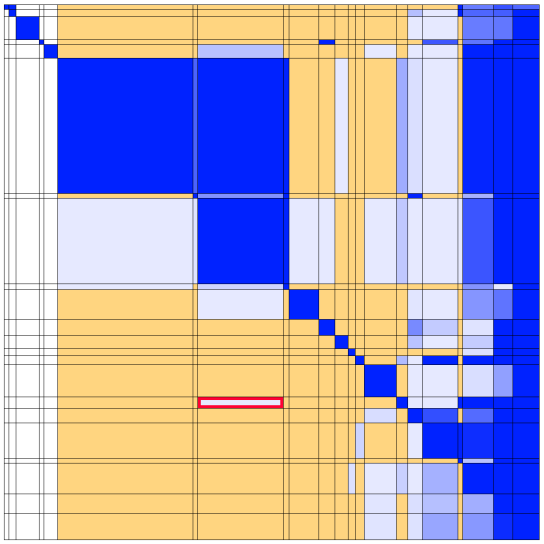
Empirically, small system parts that cause a large increase of the coupling measures are often design flaws, i.e. impair design quality (Subsections 10.3.3, 11.3.3, 12.2.3).

The limitations of the experiments have already been discussed in the respective subsections. Most notably, the experiments investigated only a small number of systems, only fairly coarse-grained subsystem decompositions, and only specific types of system parts (namely, subsystem pairs for cochange coupling and references between subsystems for d-ref coupling and t-ref coupling).

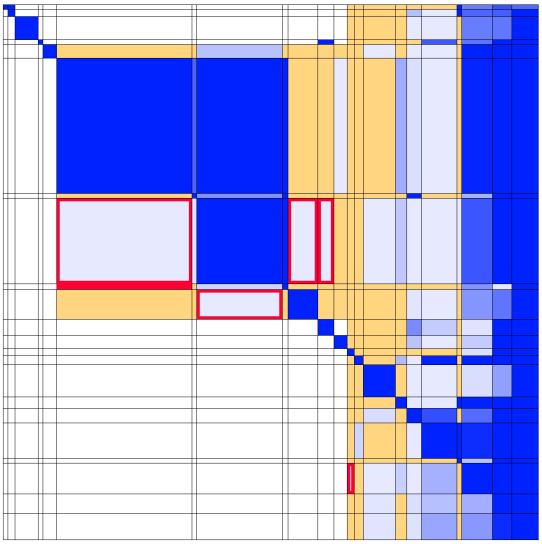
13.3 Simplification

General and valid indicators of design quality enable the automatic detection of diverse design flaws with optimization algorithms. Some of the detected design flaws are actually quite subtle (see Subsections 10.3.3, 11.3.3, and 12.2.3); for example, references from low-level responsibilities to high-level responsibilities, and the distribution of similar responsibilities over different subsystems, have been identified in graph models that do not contain any explicit information about responsibilities. And they have been identified with very simple means.

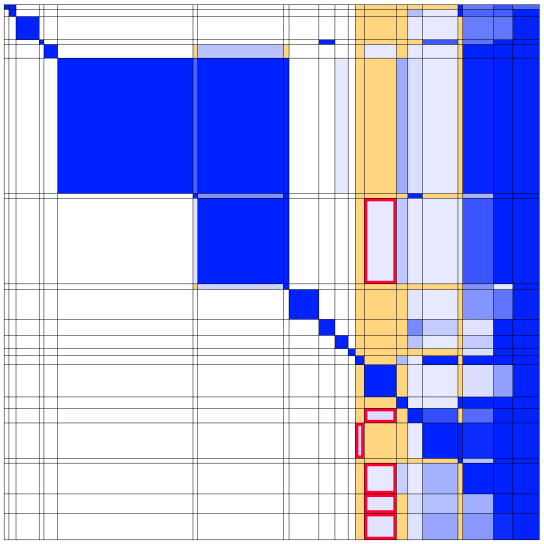
The optimization of simple measures
(the total atedge cut and the total atpair cut)
on simple system models
(cochange graphs, reference graphs, and transitive reference graphs)
can suffice to identify nonobvious and useful structure in software systems
(violations of several design principles, instances of various antipatterns,
the hidden independence behind the apparent interdependence in Figure 13.1).



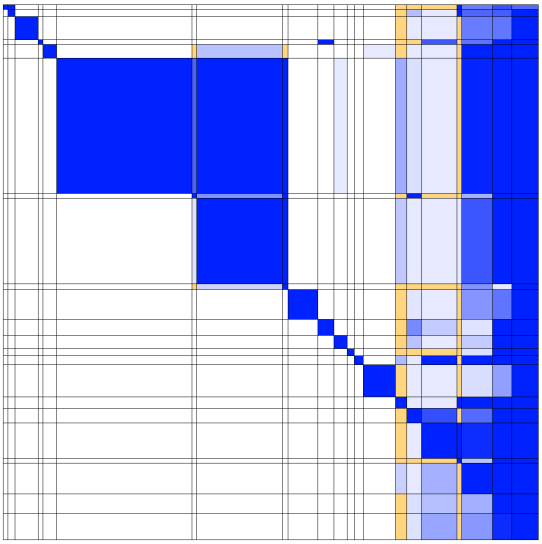
With all t-ref causes:
total t-ref coupling 3 680 346,
scaled t-ref coupling 0.894



After removing 0.02% of all t-ref causes:
total t-ref coupling 2 456 561,
scaled t-ref coupling 0.597



After removing 0.15% of all t-ref causes:
total t-ref coupling 1 808 129,
scaled t-ref coupling 0.439



After removing 0.46% of all t-ref causes:
total t-ref coupling 1 469 237,
scaled t-ref coupling 0.357

Figure 13.1: Detecting structure by optimizing the total t-ref coupling (illustrated for the software system JDK 1.4.2, see Subsection 12.2.3)

Part III

Appendices

Appendix A

Data Sets

A.1 Graphs for Part I

This appendix describes example graphs for the illustration and validation of assignment quality measures primarily in Sections 3.3, 4.4, 5.3, and 6.5. The description of each graph includes a known grouping of its vertices, which enables the external validation of graph assignments through a comparison with this grouping. Some of the graphs are directed; for the application of the assignment quality measures, the weight of the (undirected) edge between each unordered pair of vertices is set to the arithmetic mean of the weights of the corresponding two (directed) arcs.

Name	#Verts	#Edges	Edge Weight	Source
World Trade	40	1 560 arcs	$4.195 \cdot 10^{12}$	World Bank[1]
American-Eur. Trade	10	90 arcs	$1.656 \cdot 10^{12}$	World Bank[1]
US Airlines	332	2 126 edges	2 126	Pajek project[2]
ODLIS	2 896	18 238 arcs	18 238	Pajek project[2]
Southern Women	32	89 edges	89	[DGG41, p. 148]
Karate Club	34	78 edges	231	[Zac77, Figure 3]
Morse Code Confusion	36	1260 arcs	22 230	[Rot57][3]
Food Classification	45	990 edges	5 713	[HAM01, Table 5.1][4]
College Football	115	613 edges	613	Cosmus group[5]

Table A.1: Overview of example graphs

World Trade The graph models the trade between 40 countries in the year 1999. The original data set contains 66 countries; the 26 countries with the least trade,

[1] World Bank: Trade and Production Database. `http://www.worldbank.org`, Topics / Trade / Data & Statistics / Trade and Production Database. Visited August 16, 2006.

[2] Vladimir Batagelj and Andrej Mrvar: Pajek datasets. `http://vlado.fmf.uni-lj.si/pub/networks/data/`, visited August 17, 2006.

[3] Also studied e.g. in [She63], [KW78, p. 11f], [BG97, Section 4.2]

[4] Originally studied in [RM99] without a complete publication of the data.

[5] Cosmus group. `http://astro.uchicago.edu/cosmus/projects/football/`, visited August 15, 2006. Originally studied in [NG04] without a complete publication of the data.

which together contribute less the 4 percent of the total trade, were removed to improve the readability of the visualizations. Each vertex represents a country. The weight of the arc between each pair of vertices specifies the volume of the imports of the first country from the second country in US dollar.

The main factor that determines the transaction costs and thus the intensity of trade between two countries is their geographical distance. The existence of free trade agreements like the European Union and the NAFTA as another important factor rather amplifies the first factor, because such agreements mainly exist between geographically close countries.

American-European Trade This graph is identical to the World Trade graph, except that it is reduced to ten vertices representing three North American and seven European countries. It is used in Chapter 1 as small motivating example.

US Airlines The vertices of this graph represent US airports, and the unit-weight edges represent direct flights.

The probability that two airports are connected by a direct flight is strongly related to their geographical distance. Most direct flights are relatively short; only few large "hub" airports are connected by direct long-range flights.

ODLIS The vertices represent terms in the Online Dictionary of Library and Information Science (ODLIS, version December 2000), and the arcs represent hyperlinks.

A hyperlink between two terms exists if the second term is used to describe the meaning of the first term, and thus connects semantically related terms.

Southern Women The graph represents the participation of 18 women in 14 informal social events. Each woman and each event is modeled by a vertex, and each participation of a woman in an event is modeled by an edge between the corresponding vertices. The graph is bipartite: There are edges between participants and events, but not between two participants or between two events. It could be transformed into a unipartite graph where each vertex corresponds to a woman and each edge corresponds to a common participation in an event, but there are several alternative edge weighting schemes for such transformations. A more space-efficient matrix visualization of the bipartite graph could be obtained by assigning only women to the rows of the matrix and only events to the columns, but the standard visualization is used for uniformity.

Freeman [Fre03] performed a meta-analysis of 21 earlier studies of the Southern Women data which assigned the women to groups. Applying consensus analysis to combine the results of those studies, he obtained a decomposition into two groups, with the first containing Brenda, Charlotte, Eleanor, Evelyn, Frances, Laura, Pearl, Ruth, and Theresa, and the other group containing the remaining nine women. The studies show considerable disagreement about the assignment of Pearl, Olivia, and Flora. Some studies assign Pearl to the first group, some to the second group, others to no group or both groups. Olivia and Flora are assigned to the second group, considered as a separate group, or assigned to no group at all.

Karate Club Each vertex represents a member of a karate club, and the edge weight between each unordered pair of vertices specifies the number of contexts (like university classes, bars, karate tournaments) in which the corresponding two members interacted. (As usual, an edge of weight 0 is equivalent to no edge.)

The instructor of the club resigned because of conflicts with the club president over lesson fees. His supporters retaliated by leaving the club and forming a new organization with the members 1 to 9, 11 to 14, 17, 18, 20, 22. Person 9 joined this faction although he was a supporter of the club president, because he was only three weeks away from the test for black belt when the split occurred, and needed to follow the instructor to retain this rank.

Morse Code Confusion The data set consists of confusions among the auditory Morse code signals for 26 letters and 10 numerals. Morse code signals are sequences of one to five short (0.05 seconds) or long (0.15 seconds) beeps, separated by periods of silence (0.05 seconds). Subjects who did not know Morse code were required to listen to a pair of signals (separated by a quiet period of 1.4 seconds), and to state whether the two signals were the same or different. In the graph model, each vertex represents a signal. The vertex name starts with the letter or numeral corresponding to the signal, followed by dots and dashes representing short and long beeps. The weight of the arc between each ordered pair of vertices is the percentage of subjects who confused the two corresponding signals. The graph contains no loops, i.e. confusions of a signal with itself.

According to Shepard [She63], Morse code signals tend to be confused if they consist of a similar number of beeps, and have a similar fraction of short beeps.

Food Classification The data set represents the categorizations of 45 foods by 38 subjects, who were asked to sort the foods into as many categories as they wished based on perceived similarity. Each vertex corresponds to a food, and the edge between each unordered pair of vertices is weighted with the number of subjects who assigned the corresponding foods to the same category.

The original analysis of the present and additional data sets by Ross and Murphy [RM99] found that people's representations of foods are dominated by taxonomic categories like beverages, breads and grains, dairies, fruits, vegetables, and meats, but are strongly influenced by the situation where the food is eaten, e.g. as breakfast, dessert, or snack.

College Football The graph represents the schedule of Division I games of US college football in the 2000 season. Each vertex represents a football team, identified by its college name, and each unit-weight edge represents a regular season game.

The teams are divided into eleven so-called conferences, with the exception of eight teams that do not belong to any conference. The assignment of teams to conferences is shown in Table A.2, which corrects some mistakes in the original data [NG04, Figure 5]. Each team played on average about seven intra-conference games and four inter-conference games.

Atlantic Coast	**Big East**	**Big 10**
Clemson	Boston College	Illinois
Duke	Miami Florida	Indiana
Florida State	Pittsburgh	Iowa
Georgia Tech	Rutgers	Michigan
Maryland	Syracuse	Michigan State
North Carolina	Temple	Minnesota
North Carolina State	Virginia Tech	Northwestern
Virginia	West Virginia	Ohio State
Wake Forest		Penn State
		Purdue
		Wisconsin
Big 12	**Conference USA**	**Mid American**
Baylor	Alabama Birmingham	Akron
Colorado	Army	Ball State
Iowa State	Cincinnati	Bowling Green State
Kansas	East Carolina	Buffalo
Kansas State	Houston	Central Michigan
Missouri	Louisville	Eastern Michigan
Nebraska	Memphis	Kent
Oklahoma	Southern Mississippi	Marshall
Oklahoma State	Tulane	Miami Ohio
Texas		Northern Illinois
Texas A&M		Ohio
Texas Tech		Toledo
		Western Michigan
Mountain West	**Pac 10**	**SEC**
Air Force	Arizona	Alabama
Brigham Young	Arizona State	Arkansas
Colorado State	California	Auburn
Nevada Las Vegas	Oregon	Florida
New Mexico	Oregon State	Georgia
San Diego State	Southern California	Kentucky
Utah	Stanford	Louisiana State
Wyoming	UCLA	Mississippi
	Washington	Mississippi State
	Washington State	Southern Carolina
		Tennessee
		Vanderbilt
Big West	**Western Atlantic**	**No Conference**
Arkansas State	Fresno State	Central Florida
Boise State	Hawaii	Connecticut
Idaho	Nevada	Louisiana Lafayette
New Mexico State	Rice	Louisiana Monroe
North Texas	San Jose State	Louisiana Tech
Utah State	Southern Methodist	Middle Tennessee State
	Texas Christian	Navy
	Texas El Paso	Notre Dame
	Tulsa	

Table A.2: Conference membership of college football teams in the 2000 season

A.2 Software Systems for Chapter 10

This section describes the models of software systems (and of their development history) used in Chapter 10 to illustrate and evaluate cochange coupling and cochange leverage. The systems were chosen because their version repositories are freely accessible (which enables an independent verification and comparison of the results) and contain a sufficient volume of historical changes. Their predominant programming language is Java 1.4 [GJSB00].

The *software elements* are files. Because the main goals are the evaluation of design quality and the detection of design flaws, only files with program source code are included (e.g. documentation and data files are excluded). The size of each software element is 1.

The *subsystems* correspond to packages of the package hierarchy; the hierarchy level was chosen such that no package contains more than one fifth of all files or changes in the system. Packages include their subpackages, e.g. the package org.argouml.cognitive.ui is considered as part of the package org.argouml.cognitive. The size of each subsystem is the number of its contained files.

The *logical changes* were derived from repositories of the version control systems Subversion [CSFP04] and CVS [Ced05]. Users of these version control systems modify local copies of the software elements, and then commit their changes to a central repository. Basically, each set of software elements that was committed to the repository in a single transaction is considered as a logical change, as in many earlier works (e.g. [SSLM03, GJK03, ZDZ03, Ger04, HH04a, YMNCC04, ZWDZ05]). The size of each logical change is its cardinality, i.e. the number of changed software elements.

However, some commit transactions do not correspond to a coherent modification like a bug fix or a feature addition. Several heuristics for filtering out such commits have been proposed, e.g. based on the size of the transaction [SSLM03, ZW04, Ger04, YMNCC04], its textual description [HH04a], the actual modifications of the software elements [Ger04, FGP05, FG06b], or whether the transaction merges two branches of the repository [FPG03, ZW04, Ger04]. On the other hand, several transactions may be combined to a single logical change if they refer to the same modification request or bug report [FPG03, Ger04]. Because the use of complex filtering techniques with many (explicit or implicit) tunable parameters and assumptions about the repository decreases the generalizability and reproducibility of the results, the commit transactions are only filtered with respect to their size (i.e. the number of software elements) in this work. The maximum size (as the only parameter) has been set to 25; the choice is not critical, because the contribution of each logical change c to individual edge weights in the cochange graph is inversely proportional to size of c, and thus large logical changes have a small effect on individual edge weights anyway.

There are several alternative techniques for extracting logical changes, which have not been used in this work. First, even if the version control system only records changes of entire files, these changes may be mapped to more fine-grained software elements like functions by analyzing the differences between successive file

revisions [ZW04, HH04a, FGP05, ZKWZ06]. Second, logical changes can be derived not only from version repositories, but also from the navigation of users in development environments (e.g. [RM03, KM05, SES05, RLL07, ZGH07]).

	ArgoUML 0.22	Eclipse jdt.ui 3.1
Software elements	1305	1606
Subsystems	33	40 or 33 (see text)
First logical change	Sep 18 2000	May 04 2001
Last logical change	Jul 26 2006	Jun 24 2005
Logical changes	1863	3418
Total size of logical changes	9618	15549

Table A.3: Statistics of the example software systems

ArgoUML 0.22 ArgoUML is a UML (Unified Modeling Language) modeling tool. The version repository of ArgoUML has been converted from CVS to Subversion in September 2006, and the original CVS repository is no longer available. The logical changes were extracted after the conversion, from commit transactions that happened before the conversion (i.e. to the CVS repository). In contrast to Subversion, CVS records only commits of individual files, and these commits were combined to commit transactions during the conversion by the tool cvs2svn[1]. Basically, cvs2svn combines commits with the same author, the same log message, and similar time stamps; the details are described in the cvs2svn documentation, in particular in the files design-notes.txt and cvs_revision_aggregator.py.

The logical changes were extracted from the Subversion repository as follows:

- Check out a local copy of ArgoUML 0.22 from the repository:
  svn co http://argouml.tigris.org/svn/argouml/releases/VERSION_0_22 argouml --username guest
  The program svn is part of Subversion[2].
- Generate a file argouml.log with all commit transactions:
  svn log -v argouml > argouml.log
- Delete from argouml.log all files without the extension .java.
- Delete from argouml.log all files in the directories tests (133 Java files, test code), modules (13 Java files, optional modules), src/model-mdr (38 Java files, short history because the formerly separate subproject was merged into the main project only on Mar 03 2006); the remaining directories with Java files are src_new and src/model/src.
- Remove prefixes from the filenames, such that the directories correspond to the Java packages; e.g. /src_new/org/argouml/util/Tools.java is shortened to /org/argouml/util/Tools.java.
- Delete from argouml.log all transactions with less than 2 or more than 25 files.
- The remaining commit transactions in argouml.log are the logical changes.

[1] Available at http://cvs2svn.tigris.org/.
[2] Available at http://subversion.tigris.org/.

Eclipse jdt.ui 3.1 Eclipse consists of the three subprojects Platform, JDT, and PDE, which are further decomposed into so-called plug-ins. JDT is a Java IDE (integrated development environment), PDE is an IDE for Eclipse plug-ins, and Platform provides "an IDE for anything and nothing in particular" on which JDT and PDE are built. The common changes can be analyzed for each plug-in separately, because the vast majority of all commit transactions involves only files from a single plug-in. The plug-in org.eclipse.jdt.ui (shortly jdt.ui) was chosen because it contains the largest number of Java source files.

The plug-in jdt.ui contains seven pairs of an API (application programmer interface) package and a corresponding non-API (implementation detail) package, e.g. org.eclipse.jdt.ui.text and org.eclipse.jdt.internal.ui.text. Therefore, two different subsystem decompositions are used, one with 33 subsystems where these pairs are joined, and one with 40 subsystems where they are not joined. The prefix org.eclipse.jdt of the package names is always omitted.

The Eclipse project uses the version control system CVS, which only records commits of individual files. These individual commits were merged to commit transactions with the Perl script cvs2cl[3], which combines commits with the same author, the same log message, and time stamps differing by at most 180 seconds (sliding time window); the same heuristic (except for varying lengths of the time window) has been proposed and justified e.g. in [MFH02, ZW04, Ger04].

The logical changes were extracted from the CVS repository as follows:

- Set the environment variable CVSROOT to the repository location:
  set CVSROOT=:pserver:anonymous@dev.eclipse.org:/cvsroot/eclipse
- Check out a local copy of Eclipse jdt.ui 3.1 from the repository:
  cvs co -rR3_1 org.eclipse.jdt.ui
  The program cvs is part of CVS[4].
- Generate a file ChangeLog with all commit transactions:
  cvs log -rR:3_1 org.eclipse.jdt.ui | perl cvs2cl.pl --stdin
  The program perl is a Perl interpreter, e.g. ActivePerl[5].
- Delete from ChangeLog all files without the extension .java.
- Remove prefixes from the filenames, such that the directories correspond to the Java packages; e.g. core extension/org/eclipse/jdt/internal/corext/Assert.java is shortened to org/eclipse/jdt/internal/corext/Assert.java.
- Delete from ChangeLog all commit transactions with less than 2 or more than 25 files.
- The remaining commit transactions in ChangeLog are the logical changes.

[3]Available at http://www.red-bean.com/cvs2cl/.
[4]Available at http://www.nongnu.org/cvs/.
[5]Available at http://www.activestate.com/Products/ActivePerl/.

A.3 Software Systems for Chapters 11 and 12

This section describes the software systems used in Chapters 11 and 12 to illustrate and evaluate methods for detecting flawed references. The source code of the systems is freely available (which enables an independent verification and comparison of the results), reasonably well-documented, and predominantly written in the programming language Java 1.4 [GJSB00] (which enables a relatively simple and reliable extraction of references). Files that contain neither Java source code nor Java bytecode were ignored.

System	Source
ArgoUML 0.22	http://argouml.tigris.org/
Eclipse 3.1	http://www.eclipse.org/eclipse/
JDK 1.4.2_03	http://java.sun.com/j2se/1.4.2/
JHotDraw 5.4b2	http://www.jhotdraw.org/
JWAM 1.8	On request from C1 WPS GmbH, http://www.c1-wps.de/

Table A.4: Sources of the example software systems

The *software elements* are top-level classes, i.e. classes that are not declared in the body of another class. The term class includes so-called interfaces, i.e. classes without implementation. The size of each software element is 1.

The *subsystems* correspond to packages of the package hierarchy; the hierarchy level was chosen such that no package contains more than one third of all classes. Packages include their subpackages, e.g. the package java.awt.print is considered as part of the package java.awt. The only exception is Eclipse 3.1, which is hierarchically decomposed into three subprojects (PDE, JDT, and Platform) with components and plug-ins. The subsystems used in this work correspond to the components. The assignment of plug-ins to components was obtained from the online documentation at http://www.eclipse.org/eclipse/, and is shown in Table A.6. For all software systems, the size of each subsystem is the number of its contained top-level classes.

The *references* between the top-level classes were extracted with the tool Sotograph v2.110 (http://www.software-tomography.com/). In Java, a class c_1 references a class c_2 if c_1 (or one of its contained classes) invokes a method or accesses and attribute of c_2 (or one of its contained classes), or if c_2 (or one of its contained classes) appears as type in c_1 (or one of its contained classes), e.g. as superclass, in the declaration of a variable, in the signature of a method, in the invocation of a constructor, or in a cast expression. The size of each reference between two top-level classes is 1. Accordingly, the size of each reference between two subsystems is the number of references between their top-level classes.

For each ordered pair of different subsystems (s_1, s_2), it was manually assessed whether a reference from s_1 to s_2 is *acceptable*. This assessment was based on knowledge about the responsibilities of the subsystems, which was extracted from documentation. A reference from subsystem s_1 to subsystem s_2 was assessed as acceptable

System	Subsystems		Inter-Refs		Total Refs	
	Number	Size	Number	Size	Number	Size
ArgoUML 0.22	24	1397	148	4978	169	7562
Eclipse 3.1	19	11645	113	62211	132	124879
JDK 1.4.2_03	22	2161	130	11318	152	20029
JHotDraw 5.4b2	8	300	30	1123	36	1536
JWAM 1.8	11	663	32	1978	42	3390

Table A.5: Number and total size of the subsystems, of the references between different subsystems, and of all references of the example software systems

if and only if the responsibilities of s_2 are of significant value for s_1 to fulfill its responsibilities. In particular, references from general, widely reusable subsystems (like basic utilities and frameworks) to specialized, narrowly reusable subsystems (like specific tools or user interfaces) were classified as *not* acceptable. As detailed in the paragraphs on the individual systems below, sometimes the developer documentation explicitly specified which references are deemed acceptable by the system architects. Otherwise, the assessment is clearly not entirely objective.

The result of the assessment is shown in Figures A.1 to A.5. The thick horizontal lines in these figures partition the systems into *layers*. A reference from a subsystem s_1 to a subsystem s_2 is acceptable if and only if there is a path of arrows from s_1 to s_2, or if s_1 belongs to a layer (not necessarily directly) above s_2.

In Sections 11.3 and 12.2, a reference between subsystems is considered as *flawed* if and only if it is not acceptable. Strictly speaking, the assessment of acceptability does *not* identify flawed references, because it is performed for every ordered pair of different subsystems, independent of the actual existence of a reference between the subsystems.

ArgoUML 0.22 ArgoUML is a UML (Unified Modeling Language) modeling tool. The analyzed system includes the directories /argouml/src/model and /argouml/src_new of the source distribution ArgoUML-0.22-src.zip; omitted are optional extensions, e.g. for parsing or generating code in specific languages. The prefix org.argouml of the package names is always omitted. The subsystems uml.uml and uml.ui.ui contain the classes from the packages uml and uml.ui, but not from their subpackages (like uml.diagram and uml.ui.behavior), which are separate subsystems.

Chapters 4 and 5 of the "Cookbook for Developers of ArgoUML"[6] contain incomplete and partially outdated information about the responsibilities and the acceptable references of the subsystems. Further information about responsibilities is included in the package.html files and the Javadoc documentation in the source distribution. The package application was omitted because it has no clear responsibilities. On the one hand, it contains the main class which may reference all other packages; on the other hand, it contains utility classes that are needed by many other packages.

[6] http://argouml-stats.tigris.org/documentation/defaulthtml/cookbook/

Eclipse 3.1 Eclipse consists of the three subprojects Platform, JDT, and PDE. JDT is a Java IDE (integrated development environment), PDE is an IDE for Eclipse plug-ins, and Platform provides "an IDE for anything and nothing in particular" on which JDT and PDE are built. The analyzed system includes all plug-ins in the Eclipse SDK eclipse-SDK-3.1-win32.zip, except those which do not begin with org.eclipse (which are org.apache.ant, org.apache.lucene, and org.junit).

The three subprojects PDE, JDT, and Platform are decomposed into components and plug-ins; the subsystems used in this work correspond to the components. The developer documentation at http://www.eclipse.org/eclipse/ describes the responsibilities of the components, but makes no statements about acceptable references, except that the Platform should be independent of JDT and PDE. The derivation of the acceptable references from the responsibilities is described in detail in the bachelor thesis of Wenzel [Wen05].

JDK 1.4.2 JDK is the core class library for the Java programming language. The analyzed system includes the directories java and javax of the archive src.jar, which is included in the JDK distribution.

The responsibilities of the packages are described in the documentation at http://java.sun.com/j2se/1.4.2/docs/, in particular in the "Guide to Features" and the "API Specification". The package java.beans was omitted because its responsibilities are not clear. On the one hand, it provides a general component architecture that could be referenced by many other packages; on the other hand, it provides support for the persistence of many classes in java.awt and javax.swing, and thus needs to reference these high-level packages. Only few references to the packages java.net and java.security have been classified as acceptable, although these packages contain basic classes like URL and AccessController, because most of their classes are much more specific and should not be widely referenced. The new print API javax.print is explicitly required to be independent of the package java.awt (which includes the old print API java.awt.print) e.g. by the API specification of the class javax.print.DocFlavor.

JHotDraw 5.4 JHotDraw is a GUI (graphical user interface) framework for technical and structured graphics, with some example applications and applets. The analyzed system includes the entire Java source code in the distribution jhotdraw54b2.zip, except the package ch.ifa.draw.test which contains test code. The prefix ch.ifa.draw of the package names is always omitted.

The responsibilities of the packages are outlined in the file /doc/packages.html of the distribution. The intended layering is not explicitly described, but fairly clear.

JWAM 1.8 JWAM is an application framework for the tools and materials metaphor. The analyzed system includes the entire Java source code except test classes and generated RMI stubs and skeletons.

The layering of the handling, technology, and lang packages is described in the file /doc/layerswithcomponents.gif. The layering of the packages jwambeta, jwamx, and jwam is clear from the package names. References from the packages jwamdev and jwamexample to the package jwambeta.handling might be considered as acceptable; this would not significantly change the conclusions of the experiments in Sections 11.3 and 12.2.

Ant
org.eclipse.ant.core
org.eclipse.ant.ui
org.eclipse.ui.externaltools

Compare
org.eclipse.compare

CVS
org.eclipse.team.cvs.core
org.eclipse.team.cvs.ssh
org.eclipse.team.cvs.ssh2
org.eclipse.team.cvs.ui

Debug
org.eclipse.debug.core
org.eclipse.debug.ui

Help
org.eclipse.help
org.eclipse.help.appserver
org.eclipse.help.base
org.eclipse.help.ui
org.eclipse.help.webapp
org.eclipse.tomcat

JFace
org.eclipse.jface

Runtime
org.eclipse.core.boot
org.eclipse.core.commands
org.eclipse.core.expressions
org.eclipse.core.runtime
org.eclipse.core.runtime.compatibility
org.eclipse.core.variables
org.eclipse.osgi
org.eclipse.osgi.services
org.eclipse.osgi.util
org.eclipse.platform

Search
org.eclipse.search

SWT
org.eclipse.swt

Text
org.eclipse.core.filebuffers
org.eclipse.jface.text
org.eclipse.text
org.eclipse.ui.editors
org.eclipse.ui.workbench.texteditor

Team
org.eclipse.team.core
org.eclipse.team.ui

Update
org.eclipse.update.configurator
org.eclipse.update.core
org.eclipse.update.scheduler
org.eclipse.update.ui

Workbench
org.eclipse.ui
org.eclipse.ui.browser
org.eclipse.ui.cheatsheets
org.eclipse.ui.console
org.eclipse.ui.forms
org.eclipse.ui.ide
org.eclipse.ui.intro
org.eclipse.ui.presentations.r21
org.eclipse.ui.views
org.eclipse.ui.win32
org.eclipse.ui.workbench
org.eclipse.ui.workbench.compatibility

Workspace
org.eclipse.core.resources
org.eclipse.core.resources.compatibility
org.eclipse.core.resources.win32

JDT Core
org.eclipse.jdt.core

JDT Debug
org.eclipse.jdt.debug
org.eclipse.jdt.debug.ui
org.eclipse.jdt.launching

JDT UI
org.eclipse.jdt.junit
org.eclipse.jdt.junit.runtime
org.eclipse.jdt.ui
org.eclipse.ltk.core.refactoring
org.eclipse.ltk.ui.refactoring

PDE Build
org.eclipse.pde.build

PDE UI
org.eclipse.pde
org.eclipse.pde.core
org.eclipse.pde.junit.runtime
org.eclipse.pde.runtime
org.eclipse.pde.ui

Table A.6: Assignment of plug-ins to subsystems (bold) in Eclipse 3.1

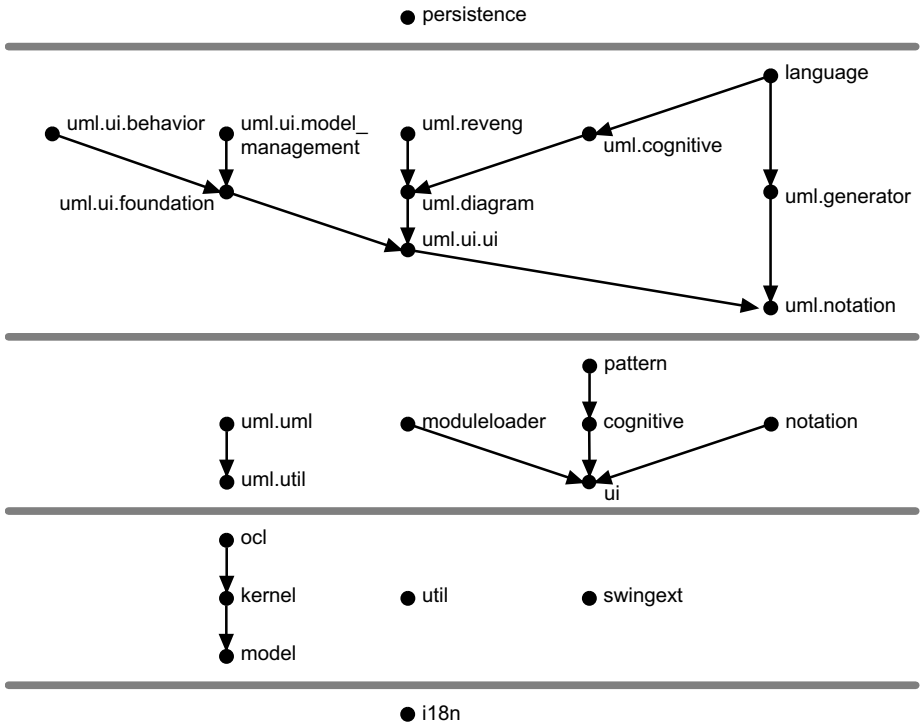


Figure A.1: Acceptable references between the subsystems of ArgoUML 0.22. A reference from a subsystem s_1 to a subsystem s_2 is acceptable if and only if there is a path of arrows from s_1 to s_2, or if s_1 belongs to a layer (not necessarily directly) above s_2.

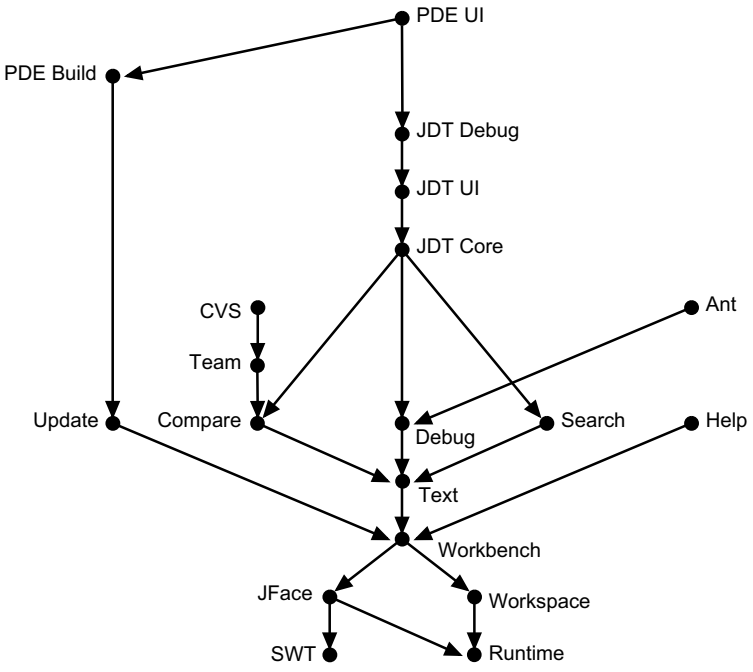


Figure A.2: Acceptable references between the subsystems of Eclipse 3.1

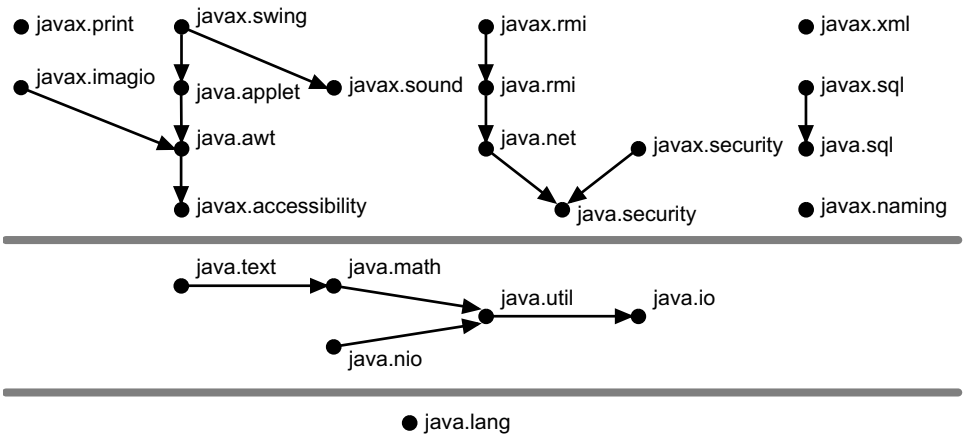


Figure A.3: Acceptable references between the subsystems of JDK 1.4.2

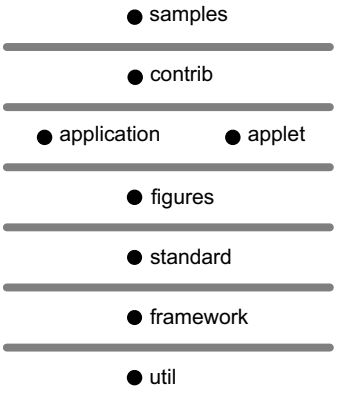


Figure A.4: Acceptable references between the subsystems of JHotDraw 5.4

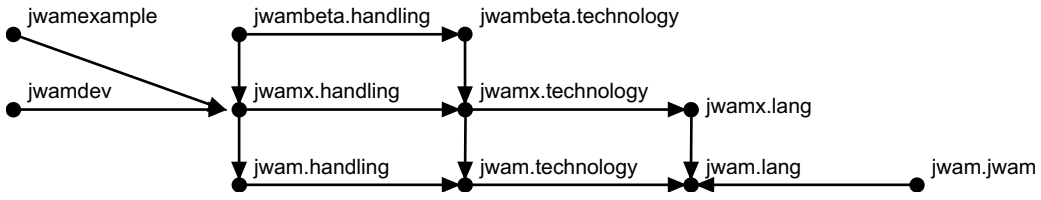


Figure A.5: Acceptable references between the subsystems of JWAM 1.8

Appendix B

Tool Support

The assignments and visualizations of real-world graphs in this work have been generated with the tool *CrocoCosmos*. CrocoCosmos was originally developed for the three-dimensional visualization of references in software systems by Lewerentz, Simon and Steinbrückner [LSS02], and was generalized and extended for the present work.

CrocoCosmos analyzes and visualizes views of subgraphs of a given hierarchically clustered graph (as defined in Subsection 2.3.2). A systematic set of navigation operations enables the user to interactively control the view and the subgraph. Firstly, the vertices and edges of the underlying graph can be aggregated along the cluster tree. For example, the user can start with a subsystem-level visualization of a software system, and then selectively zoom into subsystems of interest while keeping the global context. Secondly, arbitrary vertices and edges of the underlying graph can be filtered out, which allows, for example, to focus on the relationships between a few subsystems without being distracted by irrelevant information.

CrocoCosmos automatically computes clusterings, orderings, and layouts that reflect density and sparsity (by optimizing the quality measures introduced in this work), but also orderings and layouts that reflect other aspects of the graph, like the given cluster tree or the directions of edges. The task-oriented organization of the available layout styles, which can be extended to orderings, is described in [NL05].

CrocoCosmos not only accepts vertex weights and edge weights as input, but also computes various vertex attributes and edge attributes itself, including the degrees of vertices, the inter-density of vertices, and the number of aggregated vertices or edges of the underlying graph. When applied to models of software systems, several of these attributes coincide with common software product measures.

CrocoCosmos provides various complementary visualizations, including a two-dimensional and a three-dimensional box-and-line visualization, a matrix visualization, a tree widget showing the cluster tree, and lists of vertices and edges for the detailed examination of attribute values. The mapping of vertex and edge attributes to visual attributes in the graphical views can be controlled by the user. For example, the volume of the sphere representing a software subsystem can reflect its number of software elements, or the number of subsystems to which it is related.

Acknowledgements

This research was performed at the Software Systems Engineering Research Group of the Brandenburg Technical University, under the supervision and with the comprehensive support of Prof. Dr. Claus Lewerentz. I am grateful for the encouragement and assistance I received during my work on this thesis, and for many direct contributions to its contents including:

Discussions and Reviews Marcel Bennicke, Prof. Dr. Dirk Beyer, Marcel Damm, Martin Junghans, André Preußner, Mathias Radicke, Randolf Rotta, Dr. Heinrich Rust, Frank Steinbrückner

Tool Design and Implementation André Preußner, Frank Steinbrückner (CrocoCosmos), Michael Balzer (3D visualization frontend for CrocoCosmos)

Bachelor and Diploma Theses Oliver Huras [Hur06], Mathias Radicke [Rad05], Frank Steinbrückner [Ste05], Norman Wenzel [Wen05]

Other Student Projects Martin Junghans (layout algorithms), Randolf Rotta (clustering algorithms), Gideon Schwarz (matrix visualizations)

Data Preparation Ronny Decke, Martin Junghans, André Preußner (non-software graphs), Frank Steinbrückner (cochange graphs), Dr. Walter Bischofberger, Jan Kühl (reference graphs)

Bibliography

[AB02] Réka Albert and Albert-László Barabási. Statistical mechanics of complex networks. *Reviews of Modern Physics*, 74(1):47–97, 2002. 10

[ABH98] Jonathan E. Atkins, Erik G. Boman, and Bruce Hendrickson. A spectral algorithm for seriation and the consecutive ones problem. *SIAM Journal on Computing*, 28(1):297–310, 1998. 31

[ABK98] Mihael Ankerst, Stefan Berchtold, and Daniel A. Keim. Similarity clustering of dimensions for an enhanced visualization of multidimensional data. In *Proceedings of the IEEE Symposium on Information Visualization (InfoVis 1998)*, pages 52–59. IEEE Computer Society, 1998. 81

[ACG+99] G. Ausiello, P. Crescenzi, G. Gambosi, V. Kann, A. Marchetti-Spaccamela, and M. Protasi. *Complexity and Approximation.* Springer-Verlag, Berlin, 1999. 50, 51, 52, 53, 79, 80, 81

[ADS93] Hiralal Agrawal, Richard A. DeMillo, and Eugene H. Spafford. Debugging with dynamic slicing and backtracking. *Software – Practice and Experience*, 23(6):589–616, 1993. 148

[AH90] Phipps Arabie and Lawrence J. Hubert. The bond energy algorithm revisited. *IEEE Transactions on Systems, Man, and Cybernetics*, 20(1):268–274, 1990. 83

[AH96] Phipps Arabie and Lawrence J. Hubert. An overview of combinatorial data analysis. In P. Arabie, L. J. Hubert, and G. De Soete, editors, *Clustering and Classification*, pages 5–63. World Scientific, 1996. 29

[AHS90] Phipps Arabie, Lawrence J. Hubert, and S. Schleutermann. Blockmodels from the bond energy algorithm. *Social Networks*, 12(2):99–126, 1990. 83

[AHS96] Phipps Arabie, Lawrence J. Hubert, and Geert De Soete, editors. *Clustering and Classification.* World Scientific, Singapore, 1996. 9, 49

[AK95] Charles J. Alpert and Andrew B. Kahng. Recent directions in netlist partitioning: A survey. *Integration, the VLSI Journal*, 19(1-2):1–81, 1995. 49

[AKY05] James Abello, Stephen G. Kobourov, and Roman Yusufov. Visualizing large graphs with compound-fisheye views and treemaps. In János Pach, editor, *Proceedings of the 12th International Symposium on Graph Drawing (GD 2004)*, LNCS 3383, pages 431–441, Berlin, 2005. Springer-Verlag. 17

[AL98] Nicolas Anquetil and Timothy C. Lethbridge. Extracting concepts from file names: A new file clustering criterion. In *Proceedings of the 20th International Conference on Software Engineering (ICSE 1998)*, pages 84–93. IEEE Computer Society, 1998. 149

[AL99] Nicolas Anquetil and Timothy C. Lethbridge. Experiments with clustering as a software remodularization method. In *Proceedings of the 6th Working Conference on Reverse Engineering (WCRE 1999)*, pages 235–255. IEEE Computer Society, 1999. 149

[Ale64] Christopher Alexander. *Notes on the Synthesis of Form*. Harvard University Press, Cambridge, MA, 1964. 141

[Ans03] Kurt M. Anstreicher. Recent advances in the solution of quadratic assignment problems. *Mathematical Programming*, 97(1-2):27–42, 2003. 30

[AOH05] Taweesup Apiwattanapong, Alessandro Orso, and Mary Jean Harrold. Efficient and precise dynamic impact analysis using execute-after sequences. In *Proceedings of the 27th International Conference on Software Engineering (ICSE 2005)*, pages 432–441. ACM, 2005. 147

[AOSB06] Robert Ackland, Mathieu O'Neil, Russell Standish, and Markus Buchhorn. VOSON: A web services approach for facilitating research into online networks. In *Proceedings of the 2nd International Conference on e-Social Science*, 2006. 136

[AR98] Yonatan Aumann and Yuval Rabani. An $O(\log k)$ approximate min-cut max-flow theorem and approximation algorithm. *SIAM Journal on Computing*, 27(1):291–301, 1998. 111

[ARV04] Sanjeev Arora, Satish Rao, and Umesh V. Vazirani. Expander flows, geometric embeddings and graph partitioning. In *Proceedings of the 36th Annual ACM Symposium on Theory of Computing (STOC 2004)*, pages 222–231. ACM, 2004. 52, 53, 56

[ARV05] Giuliano Antoniol, Vincenzo Fabio Rollo, and Gabriele Venturi. Detecting groups of co-changing files in CVS repositories. In *Proceedings of the 8th International Workshop on Principles of Software Evolution (IWPSE 2005)*, pages 23–32. IEEE Computer Society, 2005. 154

[AvH04] James Abello and Frank van Ham. Matrix Zoom: A visual interface to semi-external graphs. In *Proceedings of the IEEE Symposium on Information Visualization (InfoVis 2004)*, pages 183–190. IEEE Computer Society, 2004. 17

[Axi] Axivion GmbH. Bauhaus. www.axivion.com. 199

[BAY03] James M. Bieman, Anneliese Amschler Andrews, and Helen J. Yang. Understanding change-proneness in OO software through visualization. In *Proceedings of the 11th International Workshop on Program Comprehension (IWPC 2003)*, pages 44–53. IEEE Computer Society, 2003. 154, 176

[BC00] Carliss Y. Baldwin and Kim B. Clark. *Design Rules, Volume 1: The Power of Modularity*. MIT Press, Cambridge, MA, 2000. 183

[BC01a] Ulrik Brandes and Sabine Cornelsen. Visual ranking of link structures. In F. Dehne, J.-R. Sack, and R. Tamassia, editors, *Proceedings of the 7th International Workshop on Algorithms and Data Structures (WADS 2001)*, LNCS 2125, pages 222–233, Berlin, 2001. Springer-Verlag. 109

[BC01b] Ralf Brockenauer and Sabine Cornelsen. Drawing clusters and hierarchies. In M. Kaufmann and D. Wagner, editors, *Drawing Graphs: Methods and Models*, LNCS 2025. Springer-Verlag, Berlin, 2001. 17

[BcPP98] Rainer E. Burkard, Eranda Çela, Panos M. Pardalos, and Leonidas S. Pitsoulis. The quadratic assignment problem. In D.-Z. Du and P. M. Pardalos, editors, *Handbook of Combinatorial Optimization*, volume 3, pages 241–337. Kluwer, Dordrecht, 1998. 30

[BDG+07] Ulrik Brandes, Daniel Delling, Marco Gaertler, Robert Görke, Martin Hoefer, Zoran Nikoloski, and Dorothea Wagner. On modularity clustering. *IEEE Transactions on Knowledge and Data Engineering*, 2007. Published online on 2007-09-20. 58

[BDV04] Bart Du Bois, Serge Demeyer, and Jan Verelst. Refactoring – improving coupling and cohesion of existing code. In *Proceedings of the 11th Working Conference on Reverse Engineering (WCRE 2004)*, pages 144–151. IEEE Computer Society, 2004. 200

[BDW99] Lionel C. Briand, John W. Daly, and Jürgen Wüst. A unified framework for coupling measurement in object-oriented systems. *IEEE Transactions on Software Engineering*, 25(1):91–121, 1999. 144, 200

[BDW05] Michael Burch, Stephan Diehl, and Peter Weißgerber. Visual data mining in software archives. In *Proceedings of the ACM Symposium on Software Visualization (SoftVis 2005)*, pages 37–46. ACM, 2005. 154, 177

[BE05] Ulrik Brandes and Thomas Erlebach, editors. *Network Analysis: Methodological Foundations*. LNCS 3418. Springer-Verlag, Berlin, 2005. 10

[Ber81] Jacques Bertin. *Graphics and Graphic Information Processing*. de Gruyter, Berlin, 1981. 78

[Ber01] Jacques Bertin. Matrix theory of graphics. *Information Design Journal*, 10(1):5–19, 2000/2001. 18, 78

[Ber02] Pavel Berkhin. Survey of clustering data mining techniques. Technical report, Accrue Software, Inc., 2002. 49

[BG97] Ingwer Borg and Patrick Groenen. *Modern Multidimensional Scaling: Theory and Applications*. Springer-Verlag, New York, 1997. 9, 29, 30, 78, 105, 231

[BGA06] Salah Bouktif, Yann-Gaël Guéhéneuc, and Giuliano Antoniol. Extracting change-patterns from CVS repositories. In *Proceedings of the 13th Working Conference on Reverse Engineering (WCRE 2006)*, pages 221–230. IEEE Computer Society, 2006. 154

[BGW03] Ulrik Brandes, Marco Gaertler, and Dorothea Wagner. Experiments on graph clustering algorithms. In G. Di Battista and U. Zwick, editors, *Proceedings of the 11th Annual European Symposium on Algorithm (ESA 2003)*, LNCS 2832, pages 568–579. Springer-Verlag, 2003. 52, 56

[BGW07] Ulrik Brandes, Marco Gaertler, and Dorothea Wagner. Engineering graph clustering: Models and experimental evaluation. *ACM Journal of Experimental Algorithmics*, 12:1.1, 2007. 52, 56

[BH86] Josh Barnes and Piet Hut. A hierarchical $O(N \log N)$ force-calculation algorithm. *Nature*, 324:446–449, 1986. 112

[BH99] Ivan T. Bowman and Richard C. Holt. Reconstructing ownership architectures to help understand software systems. In *Proceedings of the 7th International Workshop on Program Comprehension (IWPC 1999)*, pages 28–37. IEEE Computer Society, 1999. 149

[BH04] François Boutin and Mountaz Hascoët. Cluster validity indices for graph partitioning. In *Proceedings of the 8th International Conference on Information Visualisation (IV 2004)*, pages 376–381. IEEE Computer Society, 2004. 54, 60

[BH05] David Binkley and Mark Harman. Locating dependence clusters and dependence pollution. In *Proceedings of the IEEE International Conference on Software Maintenance (ICSM 2005)*, pages 177–186. IEEE Computer Society, 2005. 148

[BH06] Dirk Beyer and Ahmed E. Hassan. Animated visualization of software history using evolution storyboards. In *Proceedings of the 13th Working Conference on Reverse Engineering (WCRE 2006)*, pages 199–210. IEEE Computer Society, 2006. 180

[BHR96a] Michael W. Berry, Bruce Hendrickson, and Padma Raghavan. Sparse matrix reordering schemes for browsing hypertext. In J. Renegar, M. Shub, and S. Smale, editors, *The Mathematics of Numerical Analysis*, pages 99–123. American Mathematical Society, 1996. 80

[BHR96b] Franz-Josef Brandenburg, Michael Himsolt, and Christoph Rohrer. An experimental comparison of force-directed and randomized graph drawing algorithms. In F.-J. Brandenburg, editor, *Proceedings of the Symposium on Graph Drawing (GD 1995)*, LNCS 1027, pages 76–87, Berlin, 1996. Springer-Verlag. 111

[BJ05] Hamid Abdul Basit and Stan Jarzabek. Detecting higher-level similarity patterns in programs. In *Proceedings of the 10th European Software Engineering Conference held jointly with the 13th ACM SIGSOFT International Symposium on Foundations of Software Engineering (ESEC/FSE 2005)*, pages 156–165. ACM, 2005. 148

[BJDG+03] Ziv Bar-Joseph, Erik D. Demaine, David K. Gifford, Nathan Srebro, Angèle M. Hamel, and Tommi S. Jaakkola. K-ary clustering with optimal leaf ordering for gene expression data. *Bioinformatics*, 19(9):1070–1078, 2003. 84

[BJG01] Jorgen Bang-Jensen and Gregory Gutin. *Digraphs: Theory, Algorithms and Applications*. Springer-Verlag, London, 2001. 10

[BJGJ01] Ziv Bar-Joseph, David K. Gifford, and Tommi Jaakkola. Fast optimal leaf ordering for hierarchical clustering. *Bioinformatics*, 17(Suppl. 1):S22–S29, 2001. 84

[BKA+07] Stefan Bellon, Rainer Koschke, Giuliano Antoniol, Jens Krinke, and Ettore Merlo. Comparison and evaluation of clone detection tools. *IEEE Transactions on Software Engineering*, 33(9):577–591, 2007. 148

[BKPS97] Thomas Ball, Jung-Min Kim, Adam A. Porter, and Harvey P. Siy. If your version control system could talk ... In *Proceedings of the ICSE '97 Workshop on Process Modelling and Empirical Studies of Software Engineering*, 1997. 177

[BLW03] Lionel C. Briand, Yvan Labiche, and Yihong Wang. An investigation of graph-based class integration test order strategies. *IEEE Transactions on Software Engineering*, 29(7):594–607, 2003. 223

[BM06] Vladimir Batagelj and Andrej Mrvar. *Pajek Reference Manual, Version 1.17*, 2006. vlado.fmf.uni-lj.si/pub/networks/pajek/. 79

[BMMM98] William J. Brown, Raphael C. Malveau, Hays W. McCormick III, and Thomas J. Mowbray. *AntiPatterns: Refactoring Software, Architectures, and Projects in Crisis.* John Wiley & Sons, New York, NY, 1998. 142

[BMR+96] Frank Buschmann, Regine Meunier, Hans Rohnert, Peter Sommerlad, and Michael Stal. *Pattern-Oriented Software Architecture, Volume 1: A System of Patterns.* John Wiley & Sons, 1996. 207

[BMW94] Ted J. Biggerstaff, Bharat G. Mitbander, and Dallas E. Webster. Program understanding and the concept assignment problem. *Communications of the ACM*, 37(5):72–83, 1994. 148

[BN03] Mikhail Belkin and Partha Niyogi. Laplacian eigenmaps for dimensionality reduction and data representation. *Neural Computation*, 15(6):1373–1396, 2003. 31

[BN05] Dirk Beyer and Andreas Noack. Clustering software artifacts based on frequent common changes. In *Proceedings of the 13th International Workshop on Program Comprehension (IWPC 2005)*, pages 259–268. IEEE Computer Society, 2005. 154, 180

[BNL05] Dirk Beyer, Andreas Noack, and Claus Lewerentz. Efficient relational calculation for software analysis. *IEEE Transactions on Software Engineering*, 31(2):137–149, 2005. 200

[Bol98] Béla Bollobás. *Modern Graph Theory.* Springer-Verlag, New York, 1998. 10

[Boo94] Grady Booch. *Object-Oriented Analysis and Design with Applications.* Addison-Wesley, Reading, MA, 2nd edition, 1994. 206, 207

[BP01] Reinder J. Bril and André Postma. An architectural connectivity metric and its support for incremental re-architecting of large legacy systems. In *Proceedings of the 9th IEEE International Workshop on Program Comprehension (IWPC 2001)*, pages 269–280. IEEE Computer Society, 2001. 183, 199

[BP07] Ulrik Brandes and Christian Pich. Eigensolver methods for progressive multidimensional scaling of large data. In M. Kaufmann and D. Wagner, editors, *Proceedings of the 14th International Symposium on Graph Drawing (GD 2006)*, LNCS 4372, pages 42–53, Berlin, 2007. Springer-Verlag. 106

[BPS95] Stephen T. Barnard, Alex Pothen, and Horst D. Simon. A spectral algorithm for envelope reduction of sparse matrices. *Numerical Linear Algebra with Applications*, 2(4):317–334, 1995. 31, 80

[Bra01] Ulrik Brandes. Drawing on physical analogies. In M. Kaufmann and D. Wagner, editors, *Drawing Graphs: Methods and Models*, LNCS 2025, pages 71–86. Springer-Verlag, Berlin, 2001. 101

[BS94] Stephen T. Barnard and Horst D. Simon. A fast multilevel implementation of recursive spectral bisection for partitioning unstructured problems. *Concurrency: Practice & Experience*, 6(2):101–117, 1994. 31

[Bul03] P. S. Bullen. *Handbook of Means and Their Inequalities*. Kluwer, Dordrecht, 2003. 99, 100

[BW00] Adam L. Buchsbaum and Jeffery Westbrook. Maintaining hierarchical graph views. In *Proceedings of the 11th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2000)*, pages 566–575. ACM/SIAM, 2000. 17

[BW02] Ulrik Brandes and Thomas Willhalm. Visualization of bibliographic networks with a reshaped landscape metaphor. In *Proceedings of the 4th Joint Eurographics-IEEE TCVG Symposium on Visualization (VisSym 2002)*, pages 159–164, 2002. 109

[BW03] Jennifer Bevan and E. James Whitehead, Jr. Identification of software instabilities. In *Proceedings of the 10th Working Conference on Reverse Engineering (WCRE 2003)*, pages 134–145. IEEE Computer Society, 2003. 145, 154, 176

[BWL99] Lionel C. Briand, Jürgen Wüst, and Hakim Lounis. Using coupling measurement for impact analysis in object-oriented systems. In *Proceedings of the International Conference on Software Maintenance (ICSM 1999)*, pages 475–482. IEEE Computer Society, 1999. 186, 199

[BYEFN01] Reuven Bar-Yehuda, Guy Even, Jon Feldman, and Joseph Naor. Computing an optimal orientation of a balanced decomposition tree for linear arrangement problems. *Journal of Graph Algorithms and Applications*, 5(4):1–27, 2001. 84

[CC01] Trevor F. Cox and Michael A. A. Cox. *Multidimensional Scaling*. CRC Press, Boca Raton, FL, 2nd edition, 2001. 9, 29, 78, 105

[Ced05] Per Cederqvist et al. *Version Management with CVS*. Free Software Foundation, 2005. http://ximbiot.com/cvs/manual/. 235

[Cel98] Eranda Cela. *The Quadratic Assignment Problem: Theory and Algorithms*. Kluwer, Dordrecht, 1998. 30

[CF06] Deepayan Chakrabarti and Christos Faloutsos. Graph mining: Laws, generators, and algorithms. *ACM Computing Surveys*, 38(1), 2006. 10

[CGK+97] Chandra Chekuri, Andrew V. Goldberg, David R. Karger, Matthew S. Levine, and Clifford Stein. Experimental study of minimum cut algorithms. In *Proceedings of the 8th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 1997)*, pages 324–333. ACM/SIAM, 1997. 51

[Che70] Jeff Cheeger. A lower bound for the smallest eigenvalue of the laplacian. In R. C. Gunning, editor, *Problems in Analysis*, pages 195–199. Princeton University Press, Princeton, NJ, 1970. 52

[Chu97] Fan R. K. Chung. *Spectral Graph Theory*. American Mathematical Society, Providence, RI, 1997. 31, 32, 56, 83, 110

[CI90] Elliot J. Chikofsky and James H. Cross II. Reverse engineering and design recovery: A taxonomy. *IEEE Software*, 7(1):13–17, 1990. 150

[CIB] CIBIT SERC. RevJava. www.serc.nl/content/producten/download.shtml. 200

[Ciu99] Oliver Ciupke. Automatic detection of design problems in object-oriented reengineering. In *Proceedings of the 30th International Conference on Technology of Object-Oriented Languages and Systems - USA (TOOLS 1999)*, pages 18–32. IEEE Computer Society, 1999. 145, 200

[CK94] Shyam R. Chidamber and Chris F. Kemerer. A metrics suite for object oriented design. *IEEE Transactions on Software Engineering*, 20(6):476–493, 1994. 200

[CKKL02] M. Ajmal Chaumun, Hind Kabaili, Rudolf K. Keller, and François Lustman. A change impact model for changeability assessment in object-oriented software systems. *Science of Computer Programming*, 45(2):155–174, 2002. 186

[CKS05] Andreas Christl, Rainer Koschke, and Margaret-Anne D. Storey. Equipping the reflexion method with automated clustering. In *Proceedings of the 12th Working Conference on Reverse Engineering (WCRE 2005)*, pages 89–98. IEEE Computer Society, 2005. 199

[CLM96] Aniello Cimitile, Andrea De Lucia, and Malcolm Munro. A specification driven slicing process for identifying reusable functions. *Journal of Software Maintenance: Research and Practice*, 8(3):145–178, 1996. 148

[CMIBR07] Ali Civril, Malik Magdon-Ismail, and Eli Bocek-Rivele. SSDE: Fast graph drawing using sampled spectral distance embedding. In M. Kaufmann and D. Wagner, editors, *Proceedings of the 14th International Symposium on Graph Drawing (GD 2006)*, LNCS 4372, pages 30–41, Berlin, 2007. Springer-Verlag. 106

[CMM+05] Mariano Ceccato, Marius Marin, Kim Mens, Leon Moonen, Paolo Tonella, and Tom Tourwé. A qualitative comparison of three aspect mining techniques. In *Proceedings of the 13th International Workshop on Program Comprehension (IWPC 2005)*, pages 13–22. IEEE Computer Society, 2005. 148

[CMSB05] Davor Cubranic, Gail C. Murphy, Janice Singer, and Kellogg S. Booth. Hipikat: A project memory for software development. *IEEE Transactions on Software Engineering*, 31(6):446–465, 2005. 154

[Coh97] Jonathan D. Cohen. Drawing graphs to convey proximity: An incremental arrangement method. *ACM Transactions on Computer-Human Interaction*, 4(3):197–229, 1997. 106

[Com] Compuware Corp. OptimalAdvisor. javacentral.compuware.com/pasta/. 223

[CP80] J. Douglas Carroll and Sandra Pruzansky. Discrete and hybrid scaling models. In E. D. Lantermann and H. Feger, editors, *Similarity and Choice: Papers in Honour of Clyde Coombs*, pages 108–139. Huber, Bern, 1980. 5, 30

[CP96] Michael K. Coleman and D. Stott Parker. Aesthetics-based graph layout for human consumption. *Software – Practice & Experience*, 26(12):1415–1438, 1996. 111

[CP05] Gilles Caraux and Sylvie Pinloche. Permutmatrix: A graphical environment to arrange gene expression profiles in optimal linear order. *Bioinformatics*, 21(7):1280–1281, 2005. 80, 81

[CR00] Kunrong Chen and Vaclav Rajlich. Case study of feature location using dependence graph. In *Proceedings of the 8th IEEE International Workshop on Program Comprehension (IWPC 2000)*, pages 241–249, 2000. 148

[CSFP04] Ben Collins-Sussman, Brian W. Fitzpatrick, and C. Michael Pilato. *Version Control with Subversion*. O'Reilly, 2004. 235

[CSZ93] Pak K. Chan, Martine D. F. Schlag, and Jason Y. Zien. Spectral k-way ratio-cut partitioning and clustering. In *Proceedings of the 30th Design Automation Conference (DAC 1993)*, pages 749–754. ACM, 1993. 55

[CSZ94] Pak K. Chan, Martine D. F. Schlag, and Jason Y. Zien. Spectral k-way ratio-cut partitioning and clustering. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 13(9):1088–1096, 1994. 55

[Dav90] P. W. C. Davies. Why is the physical world so comprehensible? In W. H. Zurek, editor, *Proceedings of the Workshop on Complexity, Entropy, and the Physics of Information (1989)*, Santa Fe Institute Studies in the Sciences of Complexity 8, pages 61–70, Reading, MA, 1990. Addison-Wesley. 141

[DC98] Edmund Dengler and William Cowan. Human perception of laid-out graphs. In S. H. Whitesides, editor, *Proceedings of the 6th International Symposium on Graph Drawing (GD 1998)*, LNCS 1547, pages 441–443, Berlin, 1998. Springer-Verlag. 10

[DDN02] Serge Demeyer, Stéphane Ducasse, and Oscar Nierstrasz. *Object-Oriented Reengineering Patterns*. Morgan Kaufmann, San Francisco, CA, 2002. 150

[DETT99] Giuseppe Di Battista, Peter Eades, Roberto Tamassia, and Ioannis G. Tollis. *Graph Drawing: Algorithms for the Visualization of Graphs*. Prentice Hall, Upper Saddle River, NJ, 1999. 11, 101, 106

[DGG41] Allison Davis, Burleigh B. Gardner, and Mary R. Gardner. *Deep South*. The University of Chicago Press, Chicago, IL, 1941. 231

[DH73] W. E. Donath and A. J. Hoffman. Lower bounds for the partitioning of graphs. *IBM Journal of Research and Development*, 17:420–425, 1973. 31

[DH96] Ron Davidson and David Harel. Drawing graphs nicely using simulated annealing. *ACM Transactions on Graphics*, 15(4):301–331, 1996. 8, 108, 111

[DH04] Chris H. Q. Ding and Xiaofeng He. Linearized cluster assignment via spectral ordering. In *Proceedings of the 21st International Conference on Machine Learning (ICML 2004)*. ACM, 2004. 31, 32, 80, 83

[DHS00] Richard O. Duda, Peter E. Hart, and David G. Stork. *Pattern Classification*. Wiley-Interscience, New York, 2nd edition, 2000. 49

[DHZ+01] Chris H. Q. Ding, Xiaofeng He, Hongyuan Zha, Ming Gu, and Horst D. Simon. A min-max cut algorithm for graph partitioning and data clustering. In *Proceedings of the 1st International Conference on Data Mining (ICDM 2001)*, pages 107–114. IEEE Computer Society, 2001. 8, 59, 60, 61, 83

[Die00] Reinhard Diestel. *Graph Theory*. Springer-Verlag, New York, 2nd edition, 2000. 10

[Dij68] Edsger W. Dijkstra. The structure of the THE multiprogramming system. *Communications of the ACM*, 11(5):341–346, 1968. 206

[DL06] Marco D'Ambros and Michele Lanza. Reverse engineering with logical coupling. In *Proceedings of the 13th Working Conference on Reverse Engineering (WCRE 2006)*, pages 189–198. IEEE Computer Society, 2006. 145, 154, 156, 169, 176

[DLP05] Stéphane Ducasse, Michele Lanza, and Laura Ponisio. Butterflies: A visual approach to characterize packages. In *Proceedings of the 11th IEEE International Symposium on Software Metrics (METRICS 2005)*, page 7. IEEE Computer Society, 2005. 200

[DNR06] Stéphane Ducasse, Oscar Nierstrasz, and Matthias Rieger. On the effectiveness of clone detection by string matching. *Journal of Software Maintenance and Evolution: Research and Practice*, 18(1):37–58, 2006. 148

[DPS02] Josep Díaz, Jordi Petit, and Maria J. Serna. A survey of graph layout problems. *ACM Computing Surveys*, 34(3):313–356, 2002. 51, 78, 79, 80, 81, 82

[eAC94] Fernando Brito e Abreu and Rogério Carapuça. Object-oriented software engineering: Measuring and controlling the development process. In *Proceedings of the 4th International Conference on Software Quality*, 1994. 186

[Ead84] Peter Eades. A heuristic for graph drawing. *Congressus Numerantium*, 42:149–160, 1984. 8, 108

[ED01] Brian S. Everitt and Graham Dunn. *Applied Multivariate Data Analysis*. Arnold, London, 2nd edition, 2001. 9

[EGK+02] Stephen G. Eick, Todd L. Graves, Alan F. Karr, Audris Mockus, and Paul Schuster. Visualizing software changes. *IEEE Transactions on Software Engineering*, 28(4):396–412, 2002. 177

[EH00] Peter Eades and Mao Lin Huang. Navigating clustered graphs using force-directed methods. *Journal of Graph Algorithms and Applications*, 4(3):157–181, 2000. 17, 111

[EKS03] Thomas Eisenbarth, Rainer Koschke, and Daniel Simon. Locating features in source code. *IEEE Transactions on Software Engineering*, 29(3):210–224, 2003. 148

[EL96] Karin Erni and Claus Lewerentz. Applying design-metrics to object-oriented frameworks. In *Proceedings of the 3rd IEEE International Software Metrics Symposium (METRICS 1996)*, pages 64–74. IEEE Computer Society, 1996. 144, 145

[ELL01] Brian S. Everitt, Sabine Landau, and Morven Leese. *Cluster Analysis*. Arnold, London, 4th edition, 2001. 9, 10, 49

[Elm] Franz-Josef Elmer. Classycle. classycle.sourceforge.net. 199, 222

[ESBB98] Michael B. Eisen, Paul T. Spellman, Patrick O. Brown, and David Botstein. Cluster analysis and display of genome-wide expression patterns. *Proceedings of National Academy of Science of the USA*, 95:14863–14868, 1998. 84

[EW93] Stephen G. Eick and Graham J. Wills. Navigating large networks with hierarchies. In *Proceedings of IEEE Visualization 1993*, pages 204–210. IEEE Computer Society, 1993. 145, 154, 156, 176, 177

[FG04] Michael Fischer and Harald Gall. Visualizing feature evolution of large-scale software based on problem and modification report data. *Journal of Software Maintenance and Evolution: Research and Practice*, 16(6):385–403, 2004. 180

[FG06a] Michael Fischer and Harald Gall. EvoGraph: A lightweight approach to evolutionary and structural analysis of large software systems. In *Proceedings of the 13th Working Conference on Reverse Engineering (WCRE 2006)*, pages 179–188. IEEE Computer Society, 2006. 154, 176, 180

[FG06b] Beat Fluri and Harald Gall. Classifying change types for qualifying change couplings. In *Proceedings of the 14th International Conference on Program Comprehension (ICPC 2006)*, pages 35–45. IEEE Computer Society, 2006. 235

[FGP05] Beat Fluri, Harald C. Gall, and Martin Pinzger. Fine-grained analysis of change couplings. In *Proceedings of the 5th International Workshop on Source Code Analysis and Manipulation (SCAM 2005)*, pages 66–74. IEEE Computer Society, 2005. 235, 236

[Fie75] Miroslav Fiedler. A property of eigenvectors of nonnegative symmetric matrices and its application to graph theory. *Czechoslovak Mathematical Journal*, 25(100):619–633, 1975. 31, 109

[FLM95] Arne Frick, Andreas Ludwig, and Heiko Mehldau. A fast adaptive layout algorithm for undirected graphs. In R. Tamassia and I. G. Tollis, editors, *Proceedings of the DIMACS International Workshop on Graph Drawing (GD 1994)*, LNCS 894, pages 388–403, Berlin, 1995. Springer-Verlag. 111

[Fow00] Martin Fowler. *Refactoring: Improving the Design of Existing Code*. Addison-Wesley, Reading, MA, 2000. 148, 150, 151, 159, 200

[Fow01] Martin Fowler. Reducing coupling. *IEEE Software*, 18(4):102–104, 2001. 207

[FP96] Norman E. Fenton and Shari Lawrence Pfleeger. *Software Metrics: A Rigorous and Practical Approach*. ITP, London, 2nd edition, 1996. 10, 12

[FPG03] Michael Fischer, Martin Pinzger, and Harald Gall. Populating a release history database from version control and bug tracking systems. In *Proceedings of the International Conference on Software Maintenance (ICSM 2003)*, pages 23–32. IEEE Computer Society, 2003. 235

[FPSS96] Usama M. Fayyad, Gregory Piatetsky-Shapiro, and Padhraic Smyth. From data mining to knowledge discovery: An overview. In U. M. Fayyad, G. Piatetsky-Shapiro, P. Smyth, and R. Uthurusamy, editors, *Advances in Knowledge Discovery and Data Mining*, pages 1–34. AAAI/MIT Press, 1996. 9

[FR91] Thomas M. J. Fruchterman and Edward M. Reingold. Graph drawing by force-directed placement. *Software – Practice and Experience*, 21(11):1129–1164, 1991. 8, 108, 133

[Fre03] Linton C. Freeman. Finding social groups: A meta-analysis of the southern women data. In R. Breiger, K. Carley, and P. Pattison, editors, *Dynamic Social Network Modeling and Analysis*, pages 37–77. The National Academies Press, Washington, DC, 2003. 63, 232

[Gae05] Marco Gaertler. Clustering. In Ulrik Brandes and Thomas Erlebach, editors, *Network Analysis: Methodological Foundations*, LNCS 3418, pages 178–215. Springer-Verlag, Berlin, 2005. 49, 60

[GDK+07] Tudor Girba, Stephane Ducasse, Adrian Kuhn, Radu Marinescu, and Daniel Ratiu. Using concept analysis to detect co-change patterns. In *Proceedings of the 9th International Workshop on Principles of Software Evolution (IWPSE 2007)*, pages 83–89. ACM, 2007. 154

[Gel71] Alan E. Gelfand. Rapid seriation methods with archaeological applications. In F. R. Hodson, D. G. Kendall, and P. Tăutu, editors, *Mathematics in the Archaeological and Historical Sciences*, pages 186–201. Edinburgh University Press, Edinburgh, 1971. 81

[Ger04] Daniel M. German. An empirical study of fine-grained software modifications. In *Proceedings of the 20th International Conference on Software Maintenance (ICSM 2004)*, pages 316–325. IEEE Computer Society, 2004. 176, 177, 235, 237

[GFC04] Mohammad Ghoniem, Jean-Daniel Fekete, and Philippe Castagliola. A comparison of the readability of graphs using node-link and matrix-based representations. In *Proceedings of the IEEE Symposium on Information Visualization (InfoVis 2004)*, pages 17–24. IEEE Computer Society, 2004. 18

[GH61] Ralph E. Gomory and Tien Chung Hu. Multi-terminal network flows. *Journal of SIAM*, 9(4):551–570, 1961. 51, 52

[GH88] Olivier Goldschmidt and Dorit S. Hochbaum. Polynomial algorithm for the k-cut problem. In *Proceedings of the 29th Annual Symposium on Foundations of Computer Science (FOCS 1988)*, pages 444–451. IEEE Computer Society, 1988. 52

[GHJ98] Harald Gall, Karin Hajek, and Mehdi Jazayeri. Detection of logical coupling based on product release history. In *Proceedings of the International Conference on Software Maintenance (ICSM 1998)*, pages 190–197. IEEE Computer Society, 1998. 154

[GHJV95] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, Reading, MA, 1995. 142, 159, 198, 206, 209

[GJ79] Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, New York, 1979. 50, 51, 79, 80, 81, 223

[GJK03] Harald Gall, Mehdi Jazayeri, and Jacek Krajewski. CVS release history data for detecting logical couplings. In *Proceedings of the 6th International Workshop on Principles of Software Evolution (IWPSE 2003)*, pages 13–23. IEEE Computer Society, 2003. 145, 154, 156, 169, 176, 235

[GJSB00] James Gosling, Bill Joy, Guy Steele, and Gilad Bracha. *The Java Language Specification*. Addison-Wesley, 2nd edition, 2000. 235, 238

[GKS99] Jean-Francois Girard, Rainer Koschke, and Georg Schied. A metric-based approach to detect abstract data types and state encapsulations. *Automated Software Engineering*, 6(4):357–386, 1999. 149

[GL81] A. George and J. W. H. Liu. *Computer Solution of Large Sparse Positive Definite Systems*. Prentice Hall, Englewood Cliffs, NJ, 1981. 78, 80

[GL91] Keith Brian Gallagher and James R. Lyle. Using program slicing in software maintenance. *IEEE Transactions on Software Engineering*, 17(8):751–761, 1991. 148

[GL96] Gene H. Golub and Charles F. Van Loan. *Matrix Computations*. The Johns Hopkins University Press, Baltimore, MD, 3rd edition, 1996. 31

[Gor99] Allan D. Gordon. *Classification*. Chapman & Hall/CRC, 2nd edition, 1999. 9, 10, 49

[GP97] Alan George and Alex Pothen. An analysis of spectral envelope reduction via quadratic assignment problems. *SIAM Journal on Matrix Analysis and Applications*, 18(3):706–732, 1997. 80

[GP02] Gregory Gutin and Abraham P. Punnen, editors. *The Traveling Salesman Problem and Its Variations*. Kluwer, Dordrecht, 2002. 81

[GR02] Johannes Grabmeier and Andreas Rudolph. Techniques of cluster algorithms in data mining. *Data Mining and Knowledge Discovery*, 6(4):303–360, 2002. 49

[GS93] David Garlan and Mary Shaw. An introduction to software architecture. In V. Ambriola and G. Tortora, editors, *Advances in Software Engineering and Knowledge Engineering*, pages 1–39. World Scientific, Singapore, 1993. 207

[GW72] G. Gruvaeus and H. Wainer. Two additions to hierarchical cluster analysis. *The British Journal of Mathematical and Statistical Psychology*, 25(1):200–206, 1972. 84

[Hal70] Kenneth M. Hall. An r-dimensional quadratic placement algorithm. *Management Science*, 17(3):219–229, 1970. 31, 109, 133

[HAM01] Lawrence Hubert, Phipps Arabie, and Jacqueline Meulman. *Combinatorial Data Analysis: Optimization by Dynamic Programming*. SIAM, Philadelphia, PA, 2001. 231

[Hau02] Edwin Hautus. Improving Java software through package structure analysis. In *Proceedings of the 6th IASTED International Conference on Software Engineering and Applications (SEA 2002)*, 2002. 223

[HB85] David H. Hutchens and Victor R. Basili. System structure analysis: Clustering with data bindings. *IEEE Transactions on Software Engineering*, 11(8):749–757, 1985. 148

[HBV01] Maria Halkidi, Yannis Batistakis, and Michalis Vazirgiannis. On clustering validation techniques. *Journal of Intelligent Information Systems*, 17(2-3):107–145, 2001. 10, 49

[Hea] Headway Software Inc. Structure101. www.headwaysoftware.com. 199, 207, 223

[hel] hello2morrow GmbH. SonarJ. www.hello2morrow.de. 199

[HGR82] L. J. Hubert, R. G. Golledge, and G. D. Richardson. Proximity matrix reorganization and hierarchical clustering. *Environment and Planning A*, 14:195–203, 1982. 79

[HH04a] Ahmed E. Hassan and Richard C. Holt. Predicting change propagation in software systems. In *Proceedings of the 20th International Conference on Software Maintenance (ICSM 2004)*, pages 284–293. IEEE Computer Society, 2004. 156, 176, 177, 235, 236

[HH04b] Ahmed E. Hassan and Richard C. Holt. Using development history sticky notes to understand software architecture. In *Proceedings of the 12th International Workshop on Program Comprehension (IWPC 2004)*, pages 183–193. IEEE Computer Society, 2004. 199

[HHE06] Weidong Huang, Seok-Hee Hong, and Peter Eades. Layout effects on sociogram perception. In P. Healy and N. S. Nikolov, editors, *Proceedings of the 13th International Symposium on Graph Drawing (GD 2005)*, LNCS 3843, pages 262–273, Berlin, 2006. Springer-Verlag. 10

[HJ06] Stefan Hachul and Michael Jünger. An experimental comparison of fast algorithms for drawing general large graphs. In P. Healy and N. S. Nikolov, editors, *Proceedings of the 13th International Symposium on Graph Drawing (GD 2005)*, LNCS 3843, pages 235–250, Berlin, 2006. Springer-Verlag. 106, 109, 111

[HK92] Lars Hagen and Andrew B. Kahng. New spectral methods for ratio cut partitioning and clustering. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 11(9):1074–1085, 1992. 31

[HK95] Dennis J.-H. Huang and Andrew B. Kahng. When clusters meet partitions: New density-based methods for circuit decomposition. In *Proceedings of the European Design and Test Conference (EDTC 1995)*, pages 60–64. IEEE Computer Society, 1995. 55

[HKO01] Aapo Hyvärinen, Juha Karhunen, and Erkki Oja. *Independent Component Analysis*. Wiley, New York, 2001. 106

[HMS01] David Hand, Heikki Mannila, and Padhraic Smyth. *Principles of Data Mining*. MIT Press, Cambridge, MA, 2001. 9

[HRB90] Susan Horwitz, Thomas Reps, and David Binkley. Interprocedural slicing using dependence graphs. *ACM Transactions on Programming Languages and Systems*, 12(1):26–60, 1990. 147

[HS76] Lawrence Hubert and James Schultz. Quadratic assignment as a general data-analysis strategy. *British Journal of Mathematical and Statistical Psychology*, 29:190–241, 1976. 30

[HS96] Brian Henderson-Sellers. *Object-Oriented Metrics: Measures of Complexity*. Prentice Hall, Upper Saddle River, NJ, 1996. 200

[HTF01] Trevor Hastie, Robert Tibshirani, and Jerome H. Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer-Verlag, New York, NY, 2001. 49

[Hub87] Lawrence J. Hubert. *Assignment Methods in Combinatorial Data Analysis*. Marcel Dekker, New York, NY, 1987. 30

[Hur06] Oliver Huras. Layout von Matrix-Visualisierungen der Abhängigkeiten in Software-Systemen (in German). Diploma thesis, University of Technology at Cottbus, 2006. 223, 247

[IBM] IBM alphaWorks. Structural Analysis for Java. www.alphaworks.ibm.com/tech/sa4j. 200, 207, 224

[IEE90] IEEE. IEEE Std 610.12-1990: IEEE standard glossary of software engineering terminology, 1990. 159

[IHL04] Jonne Itkonen, Minna Hillebrand, and Vesa Lappalainen. Application of relation analysis to a small Java software. In *Proceedings of the 8th European Conference on Software Maintenance and Reengineering (CSMR 2004)*, pages 233–239, 2004. 169

[ISO94] ISO. ISO 8402:1994, Quality management and quality assurance – Vocabulary, 1994. 149

[ISO01] ISO. ISO/IEC 9126-1:2001, Software engineering – Product quality – Part 1: Quality model, 2001. 149

[JD88] Anil K. Jain and Richard C. Dubes. *Algorithms for Clustering Data*. Prentice Hall, Englewood Cliffs, NJ, 1988. 9, 10, 29, 49

[JJ06] Padmaja Joshi and Rushikesh K. Joshi. Microscopic coupling metrics for refactoring. In *Proceedings of the 10th European Conference on Software Maintenance and Reengineering (CSMR 2006)*, pages 145–152. IEEE Computer Society, 2006. 200

[JM92] Martin Juvan and Bojan Mohar. Optimal linear labelings and eigenvalues of graphs. *Discrete Applied Mathematics*, 36(2):153–168, 1992. 31, 79, 80

[JMF99] Anil K. Jain, M. Narasimha Murty, and Patrick J. Flynn. Data clustering: A review. *ACM Computing Surveys*, 31(3):264–323, 1999. 49

[Jol02] Ian T. Jolliffe. *Principal Component Analysis*. Springer-Verlag, Berlin, 2nd edition, 2002. 106

[JRT00] Michael Jünger, Giovanni Rinaldi, and Stefan Thienel. Practical performance of efficient minimum cut algorithms. *Algorithmica*, 26(1):172–195, 2000. 51

[JW02] Richard A. Johnson and Dean W. Wichern. *Applied Multivariate Statistical Analysis*. Prentice-Hall, Upper Saddle River, NJ, 5th edition, 2002. 9

[Kai04] Volker Kaibel. On the expansion of graphs of 0/1-polytopes. In M. Grötschel, editor, *The Sharpest Cut: The Impact of Manfred Padberg and His Work*, pages 199–216. SIAM, 2004. 53

[Kat47] Leo Katz. On the matric analysis of sociometric data. *Sociometry*, 10:233–241, 1947. 80

[KCH03] Yehuda Koren, Liran Carmel, and David Harel. Drawing huge graphs by algebraic multigrid optimization. *Multiscale Modeling and Simulation*, 1(4):645–673, 2003. 31, 109

[KDG05] Adrian Kuhn, Stéphane Ducasse, and Tudor Gîrba. Enriching reverse engineering with semantic clustering. In *Proceedings of the 12th Working Conference on Reverse Engineering (WCRE 2005)*, pages 133–142. IEEE Computer Society, 2005. 149

[KEC06] Rene Keller, Claudia M. Eckert, and P. John Clarkson. Matrices or node-link diagrams: Which visual representation is better for visualising connectivity models? *Information Visualization*, 5(1):62–76, 2006. 18

[Ken06] D. G. Kendall. Seriation. In *Encyclopedia of Statistical Sciences*. Wiley, 2006. 78

[Ker05] Joshua Kerievsky. *Refactoring to Patterns*. Addison-Wesley, Boston, MA, 2005. 150, 151

[KG06] Cory J. Kapser and Michael W. Godfrey. Supporting the analysis of clones in software systems: a case study. *Journal of Software Maintenance and Evolution: Research and Practice*, 18(2):61–82, 2006. 148

[KGH+95] David C. Kung, Jerry Gao, Pei Hsia, Jeremy Lin, and Yasufumi Toyoshima. Class firewall, test order, and regression testing of object-oriented programs. *Journal of Object Oriented Programming*, 8(2):51–65, 1995. 186

[KH02] Yehuda Koren and David Harel. A multi-scale algorithm for the linear arrangement problem. In Ludek Kucera, editor, *Proceedings of the 28th International Workshop on Graph-Theoretic Concepts in Computer Science (WG 2002)*, LNCS 2573, pages 296–309, Berlin, 2002. Springer-Verlag. 78, 85

[KK89] Tomihisa Kamada and Satoru Kawai. An algorithm for drawing general undirected graphs. *Information Processing Letters*, 31(1):7–15, 1989. 106

[KK98] George Karypis and Vipin Kumar. Multilevel k-way partitioning scheme for irregular graphs. *Journal of Parallel and Distributed Computing*, 48(1):96–129, 1998. 62

[Kle03] Jon M. Kleinberg. An impossibility theorem for clustering. In S. Becker, S. Thrun, and K. Obermayer, editors, *Advances in Neural Information Processing Systems 15 (NIPS 2002)*, pages 446–453. MIT Press, 2003. 60

[KLM+97] Gregor Kiczales, John Lamping, Anurag Mendhekar, Chris Maeda, Cristina Videira Lopes, Jean-Marc Loingtier, and John Irwin. Aspect-oriented programming. In *Proceedings of the 11th European Conference on Object Oriented Programming (ECOOP 1997)*, LNCS 1241, pages 220–242, Berlin, 1997. Springer-Verlag. 148

[KM05] Mik Kersten and Gail C. Murphy. Mylar: a degree-of-interest model for IDEs. In *Proceedings of the 4th International Conference on Aspect-Oriented Software Development (AOSD 2005)*, pages 159–168. ACM, 2005. 236

[Koh01] Teuvo Kohonen. *Self-Organizing Maps*. Springer-Verlag, Berlin, 3rd edition, 2001. 106

[Kor03] Yehuda Koren. On spectral graph drawing. In T. Warnow and B. Zhu, editors, *Proceedings of the 9th International Conference on Computing and Combinatorics (COCOON 2003)*, LNCS 2697, pages 496–508, Berlin, 2003. Springer-Verlag. 110

[Kor05] Yehuda Koren. Drawing graphs by eigenvectors: Theory and practice. *Computers and Mathematics with Applications*, 49(11-12):1867–1888, 2005. 31, 32, 109, 110

[KR90] Leonard Kaufman and Peter J. Rousseeuw. *Finding Groups in Data: An Introduction to Cluster Analysis*. John Wiley & Sons, New York, NY, 1990. 9, 49

[Kra03] Jacek Krajewski. QCR – a methodology for software evolution analysis. Master's thesis, Technical University of Vienna, 2003. 169

[KS80] Joseph B. Kruskal and Judith B. Seery. Designing network diagrams. In *Proceedings of the First General Conference on Social Graphics*, pages 22–50, Washington, DC, 1980. U. S. Bureau of the Census. 106

[KS03] Rainer Koschke and Daniel Simon. Hierarchical reflexion models. In *Proceedings of the 10th Working Conference on Reverse Engineering (WCRE 2003)*, pages 36–45. IEEE Computer Society, 2003. 199

[KVV00] Ravi Kannan, Santosh Vempala, and Adrian Vetta. On clusterings – good, bad and spectral. In *Proceedings of the 41st Annual Symposium on Foundations of Computer Science (FOCS 2000)*, pages 367–377. IEEE Computer Society, 2000. 52, 56

[KVV04] Ravi Kannan, Santosh Vempala, and Adrian Vetta. On clusterings: Good, bad and spectral. *Journal of the ACM*, 51(3):497–515, 2004. 52, 56

[KW78] Joseph B. Kruskal and Myron Wish. *Multidimensional Scaling*. Sage Publications, Beverly Hills, CA, 1978. 30, 231

[KW01] Michael Kaufmann and Dorothea Wagner, editors. *Drawing Graphs: Methods and Models*. LNCS 2025. Springer-Verlag, Berlin, 2001. 106

[Lak96] John Lakos. *Large-Scale C++ Software Design*. Addison-Wesley, Reading, MA, 1996. 181, 186, 207, 209, 210

[Lat] Lattix Inc. LDM. www.lattix.com. 199, 223

[Len74] J. K. Lenstra. Clustering a data array and the traveling salesman problem. *Operations Research*, 22(2):413–414, 1974. 83

[Ler05] Jürgen Lerner. Role assignments. In Ulrik Brandes and Thomas Erlebach, editors, *Network Analysis: Methodological Foundations*, LNCS 3418, pages 216–252. Springer-Verlag, Berlin, 2005. 11, 128

[LH89] Karl Lieberherr and Ian M. Holland. Assuring good style for object-oriented programs. *IEEE Software*, 6(5):38–48, 1989. 200

[LH93] Wei Li and Sallie M. Henry. Object-oriented metrics that predict maintainability. *Journal of Systems and Software*, 23(2):111–122, 1993. 200

[Lin73] Robert F. Ling. A computer generated aid for cluster analysis. *Communications of the ACM*, 16(6):355–361, 1973. 84

[Lis88] Barbara Liskov. Data abstraction and hierarchy. *ACM SIGPLAN Notices*, 23(5):17–34, 1988. 147

[LK94] Mark Lorenz and Jeff Kidd. *Object-Oriented Software Metrics*. Prentice Hall, Englewood Cliffs, NJ, 1994. 200

[LL07] Mircea Lungu and Michele Lanza. Exploring inter-module relationships in evolving software systems. In *Proceedings of the 11th European Conference on Software Maintenance and Reengineering (CSMR 2007)*, pages 91–100, 2007. 183

[LLR95] Nathan Linial, Eran London, and Yuri Rabinovich. The geometry of graphs and some of its algorithmic applications. *Combinatorica*, 15(2):215–245, 1995. 111

[LM06] Michele Lanza and Radu Marinescu. *Object-Oriented Metrics in Practice: Using Software Metrics to Characterize, Evaluate, and Improve the Design of Object-Oriented Systems*. Springer-Verlag, Berlin, 2006. 144, 145

[LR88] Tom Leighton and Satish Rao. An approximate max-flow min-cut theorem for uniform multicommodity flow problems with applications to approximation algorithms. In *Proceedings of the 29th Annual Symposium on Foundations of Computer Science (FOCS 1988)*, pages 422–431. IEEE Computer Society, 1988. 8, 52, 53, 56

[LR99] Tom Leighton and Satish Rao. Multicommodity max-flow min-cut theorems and their use in designing approximation algorithms. *Journal of the ACM*, 46(6):787–832, 1999. 52, 53, 56

[LR03] James Law and Gregg Rothermel. Whole program path-based dynamic impact analysis. In *Proceedings of the 25th International Conference on Software Engineering (ICSE 2003)*, pages 308–318. IEEE Computer Society, 2003. 147

[LS03] Robert Leitch and Eleni Stroulia. Assessing the maintainability benefits of design restructuring using dependency analysis. In *Proceedings of the 9th IEEE International Software Metrics Symposium (METRICS 2003)*, pages 309–322. IEEE Computer Society, 2003. 186, 210

[LSS02] Claus Lewerentz, Frank Simon, and Frank Steinbrückner. Crococosmos. In P. Mutzel, M. Jünger, and S. Leipert, editors, *Proceedings of the 9th International Symposium on Graph Drawing (GD 2001)*, LNCS 2265, pages 446–447, Berlin, 2002. Springer-Verlag. 245

[LV97] Filippo Lanubile and Giuseppe Visaggio. Extracting reusable functions by flow graph based program slicing. *IEEE Transactions on Software Engineering*, 23(4):246–259, 1997. 148

[LY05] Chun-Cheng Lin and Hsu-Chun Yen. A new force-directed graph drawing method based on edge-edge repulsion. In *Proceedings of the 9th International Conference on Information Visualisation (IV 2005)*, pages 329–334. IEEE Computer Society, 2005. 111

[Man67] Nathan Mantel. The detection of disease clustering and a generalized regression approach. *Cancer Research*, 27(2):209–220, 1967. 30

[Man06] Prem S. Mann. *Introductory Statistics*. Wiley, New York, NY, 6th edition, 2006. 9

[Mar03] Robert C. Martin. *Agile Software Development: Principles, Patterns, and Practices*. Prentice Hall, Upper Saddle River, NJ, 2003. 145, 159, 181, 186, 205, 206, 223, 224

[Mar04] Radu Marinescu. Detection strategies: Metrics-based rules for detecting design flaws. In *Proceedings of the 20th IEEE International Conference on Software Maintenance (ICSM 2004)*, pages 350–359. IEEE Computer Society, 2004. 200

[MAY03] Steven A. Morris, Benyam Asnake, and Gary G. Yen. Dendrogram seriation using simulated annealing. *Information Visualization*, 2(2):95–104, 2003. 84

[MBK97] Cathleen McGrath, Jim Blythe, and David Krackhardt. The effect of spatial arrangement on judgments and errors in interpreting graphs. *Social Networks*, 19(3):223–242, 1997. 10

[Mey97] Bertrand Meyer. *Object-Oriented Software Construction*. Prentice Hall, Upper Saddle River, NJ, 2nd edition, 1997. 147

[MFH02] Audris Mockus, Roy T. Fielding, and James D. Herbsleb. Two case studies of open source software development: Apache and

Mozilla. *ACM Transactions on Software Engineering and Methodology*, 11(3):309–346, 2002. 237

[MGL06] Naouel Moha, Yann-Gaël Guéhéneuc, and Pierre Leduc. Automatic generation of detection algorithms for design defects. In *Proceedings of the 21st International Conference on Automated Software Engineering (ASE 2006)*, pages 297–300. IEEE Computer Society, 2006. 200

[Mit97] Tom M. Mitchell. *Machine Learning*. McGraw-Hill, New York, NY, 1997. 9

[MM01a] Jonathan I. Maletic and Andrian Marcus. Supporting program comprehension using semantic and structural information. In *Proceedings of 23rd International Conference on Software Engineering (ICSE 2001)*, pages 103–112. IEEE Computer Society, 2001. 149

[MM01b] Andrian Marcus and Jonathan I. Maletic. Identification of high-level concept clones in source code. In *Proceedings of 16th IEEE International Automated Software Engineering Conference (ASE 2001)*, pages 107–114. IEEE Computer Society, 2001. 148

[MM06] Brian S. Mitchell and Spiros Mancoridis. On the automatic modularization of software systems using the Bunch tool. *IEEE Transactions on Software Engineering*, 32(3):193–208, 2006. 54

[MMB05] Chris Muelder, Kwan-Liu Ma, and Tony Bartoletti. A visualization methodology for characterization of network scans. In *Proceedings of the IEEE Workshop on Visualization for Computer Security (VizSec'05)*. IEEE Computer Society, 2005. 136

[MMR+98] S. Mancoridis, B. S. Mitchell, C. Rorres, Y. Chen, and E. R. Gansner. Using automatic clustering to produce high-level system organizations of source code. In *Proceedings of the 6th IEEE International Workshop on Program Comprehension (IWPC 1998)*, pages 45–52. IEEE Computer Society, 1998. 54

[MNS95] Gail C. Murphy, David Notkin, and Kevin J. Sullivan. Software reflexion models: Bridging the gap between source and high-level models. In *Proceedings of the 3rd ACM SIGSOFT Symposium on Foundations of Software Engineering (FSE 1995)*, pages 18–28. ACM, 1995. 199

[MNS01] Gail C. Murphy, David Notkin, and Kevin J. Sullivan. Software reflexion models: Bridging the gap between design and implementation. *IEEE Transactions on Software Engineering*, 27(4):364–380, 2001. 199

[Moh89] Bojan Mohar. Isoperimetric numbers of graphs. *Journal of Combinatorial Theory, Series B*, 47(3):274–291, 1989. 52

[MRB+05] Andrian Marcus, Vaclav Rajlich, Joseph Buchta, Maksym Petrenko, and Andrey Sergeyev. Static techniques for concept location in object-oriented code. In *Proceedings of the 13th International Workshop on Program Comprehension (IWPC 2005)*, pages 33–42. IEEE Computer Society, 2005. 148

[MRB06] Alan MacCormack, John Rusnak, and Carliss Y. Baldwin. Exploring the structure of complex software designs: An empirical study of open source and proprietary code. *Management Science*, 52(9):1015–1030, 2006. 183, 210

[MRS96] Burkhard Monien, Friedhelm Ramme, and Helmut Salmen. A parallel simulated annealing algorithm for generating 3d layouts of undirected graphs. In F.-J. Brandenburg, editor, *Proceedings of the Symposium on Graph Drawing (GD 1995)*, LNCS 1027, pages 396–408, Berlin, 1996. Springer-Verlag. 111

[MS90] David W. Matula and Farhad Shahrokhi. Sparsest cuts and bottlenecks in graphs. *Discrete Applied Mathematics*, 27(1-2):113–123, 1990. 8, 53

[MS00] Erkki Mäkinen and Harri Siirtola. Reordering the reorderable matrix as an algorithmic problem. In Michael Anderson, Peter Cheng, and Volker Haarslev, editors, *Proceedings of the 1st International Symposium on Theory and Application of Diagrams (Diagrams 1999)*, LNCS 1889, pages 453–467, Berlin, 2000. Springer-Verlag. 84

[MSRM04] Andrian Marcus, Andrey Sergeyev, Václav Rajlich, and Jonathan I. Maletic. An information retrieval approach to concept location in source code. In *Proceedings of 11th Working Conference on Reverse Engineering (WCRE 2004)*, pages 214–223. IEEE Computer Society, 2004. 149

[MSW72] William T. McCormick Jr., Paul J. Schweitzer, and Thomas W. White. Problem decomposition and data reorganization by a clustering technique. *Operations Research*, 20(5):993–1009, 1972. 83

[MT04] Tom Mens and Tom Tourwé. A survey of software refactoring. *IEEE Transactions on Software Engineering*, 30(2):126–139, 2004. 150

[MT07a] Hayden Melton and Ewan Tempero. The CRSS metric for package design quality. In *Proceedings of the 30th Australasian Conference on Computer Science (ACSC 2007)*, pages 201–210. Australian Computer Society, 2007. 207, 210

[MT07b] Hayden Melton and Ewan Tempero. JooJ: Real-time support for avoiding cyclic dependencies. In *Proceedings of the 30th Australasian Conference on Computer Science (ACSC 2007)*, pages 87–95. Australian Computer Society, 2007. 223

[MW01] Audris Mockus and David M. Weiss. Globalization by chunking: A quantitative approach. *IEEE Software*, 18(2):30–37, 2001. 154, 176

[MWD99] Kim Mens, Roel Wuyts, and Theo D'Hondt. Declaratively codifying software architectures using virtual software classifications. In *Proceedings of the 29th International Conference on Technology of Object-Oriented Languages and Systems - Europe (TOOLS 1999)*, pages 33–45. IEEE Computer Society, 1999. 199

[MX03] Marina Meila and Liang Xu. Multiway cuts and spectral clustering. Technical report, University of Washington, Department of Statistics, 2003. 57

[Nei96] James M. Neighbors. Finding reusable software components in large systems. In *Proceedings of the 3rd Working Conference on Reverse Engineering (WCRE 1996)*, pages 2–10. IEEE Computer Society, 1996. 149

[NEK94] Jim Q. Ning, Andre Engberts, and Wojtech Kozaczynski. Automated support for legacy code understanding. *Communications of the ACM*, 37(5):50–57, 1994. 148

[New03] Mark E. J. Newman. The structure and function of complex networks. *SIAM Review*, 45(2):167–256, 2003. 3, 10

[New04a] M. E. J. Newman. Analysis of weighted networks. *Physical Review E*, 70:056131, 2004. 8, 58, 61, 136

[New04b] M. E. J. Newman. Detecting community structure in networks. *The European Physical Journal B*, 38(2):321–330, 2004. 49

[New06] M. E. J. Newman. Finding community structure in networks using the eigenvectors of matrices. *Physical Review E*, 74:036104, 2006. 8, 58, 61, 136

[NG04] M. E. J. Newman and M. Girvan. Finding and evaluating community structure in networks. *Physical Review E*, 69:026113, 2004. 58, 136, 231, 233

[NH01] Laura R. Novick and Sean M. Hurley. To matrix, network, or hierarchy: That is the question. *Cognitive Psychology*, 42(2):158–216, 2001. 18

[NL05] Andreas Noack and Claus Lewerentz. A space of layout styles for hierarchical graph models of software systems. In *Proceedings of the 2nd ACM Symposium on Software Visualization (SoftVis 2005)*, pages 155–164. ACM, 2005. 17, 111, 245

[Noa04] Andreas Noack. An energy model for visual graph clustering. In G. Liotta, editor, *Proceedings of the 11th International Symposium on Graph Drawing (GD 2003)*, LNCS 2912, pages 425–436, Berlin, 2004. Springer-Verlag. 105, 107, 136

[Noa06] Andreas Noack. Energy-based clustering of graphs with nonuniform degrees. In P. Healy and N. S. Nikolov, editors, *Proceedings of the 13th International Symposium on Graph Drawing (GD 2005)*, LNCS 3843, pages 309–320, Berlin, 2006. Springer-Verlag. 110

[NS05] Marc Nunkesser and Daniel Sawitzki. Blockmodels. In Ulrik Brandes and Thomas Erlebach, editors, *Network Analysis: Methodological Foundations*, LNCS 3418, pages 253–292. Springer-Verlag, Berlin, 2005. 11

[OAH03] Alessandro Orso, Taweesup Apiwattanapong, and Mary Jean Harrold. Leveraging field data for impact analysis and regression testing. In *Proceedings of the 9th European Software Engineering Conference held jointly with the 11th ACM SIGSOFT International Symposium on Foundations of Software Engineering (ESEC/FSE 2003)*, pages 128–137. ACM, 2003. 147, 148

[OC06] Mark O'Keeffe and Mel Ó Cinnéide. Search-based software maintenance. In *Proceedings of the 10th European Conference on Software Maintenance and Reengineering (CSMR 2006)*, pages 249–258. IEEE Computer Society, 2006. 145

[OMB03] Ciaran O'Reilly, Philip J. Morrow, and David W. Bustard. Lightweight prevention of architectural erosion. In *Proceedings of the 6th International Workshop on Principles of Software Evolution (IWPSE 2003)*, pages 59–64. IEEE Computer Society, 2003. 199

[OO84] Karl J. Ottenstein and Linda M. Ottenstein. The program dependence graph in a software development environment. In *Proceedings of the ACM SIGSOFT/SIGPLAN Software Engineering Symposium on Practical Software Development Environments (SDE 1984)*, pages 177–184, 1984. 147

[Opd92] William F. Opdyke. *Refactoring Object-Oriented Frameworks*. PhD thesis, University of Illinois at Urbana-Champaign, 1992. 150

[Par72] David L. Parnas. On the criteria to be used in decomposing systems into modules. *Communications of the ACM*, 15(12):1053–1058, 1972. 141, 159, 206

[Par79] David L. Parnas. Designing software for ease of extension and contraction. *IEEE Transactions on Software Engineering*, 5(2):128–138, 1979. 207

[Pet03] Jordi Petit. Experiments on the minimum linear arrangement problem. *Journal of Experimental Algorithmics*, 8, 2003. 78

[PGM+07] Denys Poshyvanyk, Yann-Gaël Guéhéneuc, Andrian Marcus, Giuliano Antoniol, and Václav Rajlich. Feature location using probabilistic ranking of methods based on execution scenarios and information retrieval. *IEEE Transactions on Software Engineering*, 33(6):420–431, 2007. 148, 149

[PHB00] Jan Puzicha, Thomas Hofmann, and Joachim M. Buhmann. A theory of proximity based clustering: structure detection by optimization. *Pattern Recognition*, 33(4):617–634, 2000. 60

[Pis84] Sergio Pissanetzky. *Sparse Matrix Technology*. Academic Press, London, 1984. 78, 80

[PJ00] Meilir Page-Jones. *Fundamentals of Object-Oriented Design in UML*. Dorset House, New York, NY, 2000. 210

[Pos03] André Postma. A method for module architecture verification and its application on a large component-based system. *Information and Software Technology*, 45(4):171–194, 2003. 199

[Pot97] Alex Pothen. Graph partitioning algorithms with applications to scientific computing. In D. E. Keyes, A. Sameh, and V. Venkatakrishnan, editors, *Parallel Numerical Algorithms*, pages 323–368. Kluwer, 1997. 49

[PS76] Christos H. Papadimitriou and Kenneth Steiglitz. Some complexity results for the Traveling Salesman Problem. In *Proceedings of the 8th Annual ACM Symposium on Theory of Computing (STOC 1976)*, pages 1–9. ACM, 1976. 81

[PST00] Tomaz Pisanski and John Shawe-Taylor. Characterizing graph drawing with eigenvectors. *Journal of Chemical Information and Computer Sciences*, 40(3):567–571, 2000. 31

[Rad05] Mathias Radicke. Minimierung transitiver Abhängigkeiten in Software-Systemen (in German). Diploma thesis, University of Technology at Cottbus, 2005. 210, 247

[Rai02] Marcus Raitner. HGV: A library for hierarchies, graphs, and views. In M. T. Goodrich and S. G. Kobourov, editors, *Proceedings of the 10th International Symposium on Graph Drawing (GD 2002)*, LNCS 2528, pages 236–243, Berlin, 2002. Springer-Verlag. 17

[RB06] Jörg Reichardt and Stefan Bornholdt. Statistical mechanics of community detection. *Physical Review E*, 74:016110, 2006. 58, 61

[RD04] Filip Van Rysselberghe and Serge Demeyer. Studying software evolution information by visualizing the change history. In *Proceedings of the International Conference on Software Maintenance (ICSM 2004)*, pages 328–337. IEEE Computer Society, 2004. 154

[RFG05] Jacek Ratzinger, Michael Fischer, and Harald Gall. EvoLens: Lens-view visualizations of evolution data. In *Proceedings of the 8th International Workshop on Principles of Software Evolution (IWPSE 2005)*, pages 103–112. IEEE Computer Society, 2005. 162, 176

[Rie96] Arthur J. Riel. *Object-Oriented Design Heuristics*. Addison-Wesley, Reading, MA, 1996. 159, 200, 207, 224

[RL06] Stefan Roock and Martin Lippert. *Refactoring in Large Software Projects: Performing Complex Restructurings Successfully*. John Wiley & Sons, 2006. 150, 151, 181, 186, 207, 222

[RLL07] Romain Robbes, Michele Lanza, and Mircea Lungu. An approach to software evolution based on semantic change. In M. Dwyer and A. Lopes, editors, *Proceedings of the 10th International Conference on Fundamental Approaches to Software Engineering (FASE 2007)*, LNCS 4422, pages 27–41. Springer, 2007. 236

[RM99] Brian H. Ross and Gregory L. Murphy. Food for thought: Cross-classification and category organization in a complex real-world domain. *Cognitive Psychology*, 38(4):495–553, 1999. 231, 233

[RM03] Martin P. Robillard and Gail C. Murphy. Automatically inferring concern code from program investigation activities. In *Proceedings of the 18th International Conference on Automated Software Engineering (ASE 2003)*, pages 225–234. IEEE Computer Society, 2003. 236

[Rob79] Fred S. Roberts. *Measurement Theory with Applications to Decision-making, Utility, and the Social Sciences*. Addison-Wesley, Reading, MA, 1979. 10, 12

[Rob05] Martin P. Robillard. Automatic generation of suggestions for program investigation. In *Proceedings of the 10th European Software Engineering Conference held jointly with the 13th ACM SIGSOFT*

International Symposium on Foundations of Software Engineering (ESEC/FSE 2005), pages 11–20. ACM, 2005. 186

[Rot57] E. Z. Rothkopf. A measure of stimulus similarity and errors in some paired-associate learning tasks. *Journal of Experimental Psychology*, 53(2):94–101, 1957. 231

[RR98] Satish Rao and Andréa W. Richa. New approximation techniques for some ordering problems. In *Proceedings of the 9th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 1998)*, pages 211–218. ACM/SIAM, 1998. 79, 81

[RR04] Satish Rao and Andréa W. Richa. New approximation techniques for some linear ordering problems. *SIAM Journal on Computing*, 34(2):388–404, 2004. 79, 81

[RS96] Vaclav Rajlich and João H. Silva. Evolution and reuse of orthogonal architecture. *IEEE Transactions on Software Engineering*, 22(2):153–157, 1996. 207

[RS98] Tom Roxborough and Arunabha Sen. Graph clustering using multiway ratio cut. In G. Di Battista, editor, *Proceedings of the 5th International Symposium on Graph Drawing (GD 1997)*, LNCS 1353, pages 291–296, Berlin, 1998. Springer-Verlag. 54

[RST+04] Xiaoxia Ren, Fenil Shah, Frank Tip, Barbara G. Ryder, and Ophelia Chesley. Chianti: A tool for change impact analysis of Java programs. In *Proceedings of the 19th Annual ACM SIGPLAN Conference on Object-oriented Programming, Systems, Languages, and Applications (OOPSLA 2004)*, pages 432–448. ACM, 2004. 147

[RVP06] Aoun Raza, Gunther Vogel, and Erhard Plödereder. Bauhaus - a tool suite for program analysis and reverse engineering. In L. Pinho and M. Harbour, editors, *Proceedings of the 11th International Conference on Reliable Software Technologies (Ada-Europe 2006)*, LNCS 4006, pages 71–82, Berlin, 2006. Springer-Verlag. 199

[SAP+06] Catherine Stringfellow, C. D. Amory, Dileep Potnuri, Anneliese Amschler Andrews, and Manfred Georg. Comparison of software architecture reverse engineering methods. *Information & Software Technology*, 48(7):484–497, 2006. 176

[SBBP05] Olaf Seng, Markus Bauer, Matthias Biehl, and Gert Pache. Search-based improvement of subsystem decompositions. In *Proceedings of the Conference on Genetic and Evolutionary Computation (GECCO 2005)*, pages 1045–1051. ACM, 2005. 145

[SC96] Geert De Soete and J. Douglas Carroll. Tree and other network models for representing proximity data. In P. Arabie, L. J. Hubert, and G. De Soete, editors, *Clustering and Classification*, pages 157–197. World Scientific, 1996. 29, 30

[Sco00] John Scott. *Social Network Analysis: A Handbook*. Sage, London, 2nd edition, 2000. 10, 11

[SES05] Janice Singer, Robert Elves, and Margaret-Anne Storey. NavTracks: Supporting navigation in software maintenance. In *Proceedings of the International Conference on Software Maintenance (ICSM 2005)*, pages 325–334. IEEE Computer Society, 2005. 236

[SG76] Sartaj Sahni and Teofilo Gonzales. P-complete approximation problems. *Journal of the ACM*, 23(3):555–565, 1976. 30

[SG03] Alexander Strehl and Joydeep Ghosh. Relationship-based clustering and visualization for high-dimensional data mining. *INFORMS Journal on Computing*, 15(2):208–230, 2003. 84

[SGCH01] Kevin J. Sullivan, William G. Griswold, Yuanfang Cai, and Ben Hallen. The structure and value of modularity in software design. In *Proceedings of the 8th European Software Engineering Conference held jointly with the 9th ACM SIGSOFT International Symposium on Foundations of Software Engineering (ESEC/FSE 2001)*, pages 99–108. ACM, 2001. 183

[SGM00] Houari A. Sahraoui, Robert Godin, and Thierry Miceli. Can metrics help to bridge the gap between the improvement of OO design quality and its automation? In *Proceedings of the International Conference on Software Maintenance (ICSM 2000)*, pages 154–162. IEEE Computer Society, 2000. 200

[She63] Roger N. Shepard. Analysis of proximities as a technique for the study of information processing in man. *Human Factors*, 5:33–48, February 1963. 231, 233

[Sim62] Herbert A. Simon. The architecture of complexity. *Proceedings of the American Philosophical Society*, 106(6):467–482, 1962. 3, 141

[SJSJ05] Neeraj Sangal, Ev Jordan, Vineet Sinha, and Daniel Jackson. Using dependency models to manage complex software architecture. In *Proceedings of the 20th Annual ACM SIGPLAN Conference on Object-oriented Programming, Systems, Languages, and Applications (OOPSLA 2005)*, pages 167–176. ACM, 2005. 183, 199, 223

[SLT06] Mazeiar Salehie, Shimin Li, and Ladan Tahvildari. A metric-based heuristic framework to detect object-oriented design flaws. In *Proceedings of the 14th International Conference on Program Comprehension (ICPC 2006)*, pages 159–168. IEEE Computer Society, 2006. 200

[SM97] Jianbo Shi and Jitendra Malik. Normalized cuts and image segmentation. In *Proceedings of the Conference on Computer Vision and Pattern Recognition (CVPR 1997)*, pages 731–737. IEEE Computer Society, 1997. 55, 57, 58

[SM00] Jianbo Shi and Jitendra Malik. Normalized cuts and image segmentation. *IEEE Transaction on Pattern Analysis and Machine Intelligence*, 22(8):888–905, 2000. 8, 31, 32, 55, 57, 58, 83

[SM05] Harri Siirtola and Erkki Mäkinen. Constructing and reconstructing the reorderable matrix. *Information Visualization*, 4(1):32–48, 2005. 84

[SMC74] W. P. Stevens, G. J. Myers, and L. L. Constantine. Structured design. *IBM Systems Journal*, 13(2):115–139, 1974. 141

[Sne57] P. H. A. Sneath. The application of computers to taxonomy. *Journal of General Microbiology*, 17:201–226, 1957. 84

[Sof] Software-Tomography GmbH. Sotograph. www.software-tomography.com. 199, 200, 207, 209, 224

[SR03] Lawrence K. Saul and Sam T. Roweis. Think globally, fit locally: Unsupervised learning of low dimensional manifolds. *Journal of Machine Learning Research*, 4:119–155, 2003. 106

[SRB06] Ilya Safro, Dorit Ron, and Achi Brandt. Graph minimum linear arrangement by multilevel weighted edge contractions. *Journal of Algorithms*, 60(1):24–41, 2006. 78

[SS00] Sudeep Sarkar and Padmanabhan Soundararajan. Supervised learning of large perceptual organization: Graph spectral partitioning and learning automata. *IEEE Transaction on Pattern Analysis and Machine Intelligence*, 22(5):504–525, 2000. 55

[SS06] Jirí Síma and Satu Elisa Schaeffer. On the NP-completeness of some graph cluster measures. In *Proceedings of the 32nd Conference on Current Trends in Theory and Practice of Computer Science (SOFSEM 2006)*, LNCS 3831, pages 530–537, Berlin, 2006. Springer-Verlag. 56

[SSB06] Olaf Seng, Johannes Stammel, and David Burkhart. Search-based determination of refactorings for improving the class structure of object-oriented systems. In *Proceedings of the Conference on Genetic and*

Evolutionary Computation (GECCO 2006), pages 1909–1916. ACM, 2006. 145

[SSC96] Mohlalefi Sefika, Aamod Sane, and Roy H. Campbell. Monitoring compliance of a software system with its high-level design models. In *Proceedings of the 18th International Conference on Software Engineering (ICSE 1996)*, pages 387–396. IEEE Computer Society, 1996. 199, 207

[SSL01] Frank Simon, Frank Steinbrückner, and Claus Lewerentz. Metrics based refactoring. In *Proceedings of the 5th European Conference on Software Maintenance and Reengineering (CSMR 2002)*, pages 30–38. IEEE Computer Society, 2001. 200

[SSLM03] Jelber Sayyad-Shirabad, Timothy C. Lethbridge, and Stan Matwin. Mining the maintenance history of a legacy software system. In *Proceedings of the 26th International Conference on Software Maintenance (ICSM 2003)*, pages 95–104. IEEE Computer Society, 2003. 235

[SSM06] Frank Simon, Olaf Seng, and Thomas Mohaupt. *Code-Quality-Management: Technische Qualität industrieller Softwaresysteme transparent und vergleichbar gemacht (in German).* dpunkt.verlag, Heidelberg, 2006. 144, 145, 207, 209

[Ste46] S. S. Stevens. On the theory and scales of measurement. *Science*, 103(2684):677–680, 1946. 10

[Ste81] Donald V. Steward. The design structure system: A method for managing the design of complex systems. *IEEE Transactions on Engineering Management*, 28(3):71–74, 1981. 183

[Ste05] Frank Steinbrückner. Analyse der Änderungskopplungen in Softwaresystemen (in German). Diploma thesis, University of Technology at Cottbus, 2005. 156, 247

[Str01] Steven H. Strogatz. Exploring complex networks. *Nature*, 410:268–276, 2001. 10

[SV91] Huzur Saran and Vijay V. Vazirani. Finding k-cuts within twice the optimal. In *Proceedings of the 32nd Annual Symposium on Foundations of Computer Science (FOCS 1991)*, pages 743–751, 1991. 52

[SV95] Huzur Saran and Vijay V. Vazirani. Finding k-cuts within twice the optimal. *SIAM Journal on Computing*, 24(1):101–118, 1995. 52

[SZG+96] Doug Schaffer, Zhengping Zuo, Saul Greenberg, Lyn Bartram, John Dill, Shelli Dubs, and Mark Roseman. Navigating hierarchically clustered networks through fisheye and full-zoom methods. *ACM Transactions on Computer-Human Interaction*, 3(2):162–188, 1996. 17

[TCS05] Nikolaos Tsantalis, Alexander Chatzigeorgiou, and George Stephanides. Predicting the probability of change in object-oriented systems. *IEEE Transactions on Software Engineering*, 31(7):601–614, 2005. 186

[Tip95] Frank Tip. A survey of program slicing techniques. *Journal of Programming Languages*, 3(3):121–189, 1995. 147

[TK04] Ladan Tahvildari and Kostas Kontogiannis. Improving design quality using meta-pattern transformations: a metric-based approach. *Journal of Software Maintenance and Evolution: Research and Practice*, 16(4-5):331–361, 2004. 200

[Ton03] Paolo Tonella. Using a concept lattice of decomposition slices for program understanding and impact analysis. *IEEE Transactions on Software Engineering*, 29(6):495–509, 2003. 148

[Tuf83] Edward R. Tufte. *The Visual Display of Quantitative Information.* Graphics Press, Cheshire, CT, 1983. 19

[Tuk77] John W. Tukey. *Exploratory Data Analysis.* Addison-Wesley, Reading, MA, 1977. 9

[Val] Valtech GmbH. dependometer. sourceforge.net/projects/dependometer. 199

[vD00] Stijn Marinus van Dongen. *Graph Clustering by Flow Simulation.* PhD thesis, Universiteit Utrecht, 2000. 52

[vGVvdW04] Rob van Gulik, Fabio Vignoli, and Huub van de Wetering. Mapping music in the palm of your hand, explore and discover your collection. In *Proceedings of the 5th International Conference on Music Information Retrieval (ISMIR 2004)*, 2004. 136

[vH03] Frank van Ham. Using multilevel call matrices in large software projects. In *Proceedings of the IEEE Symposium on Information Visualization (InfoVis 2003)*, pages 227–232. IEEE Computer Society, 2003. 183

[vHvW04] Frank van Ham and Jarke J. van Wijk. Interactive visualization of small world graphs. In *Proceedings of the IEEE Symposium on Information Visualization (InfoVis 2004)*, pages 199–206, 2004. 17, 136

[vR79] C. J. van Rijsbergen. *Information Retrieval*. Butterworths, London, 2nd edition, 1979. 187

[Wat04] Duncan J. Watts. The "new" science of networks. *Annual Review of Sociology*, 30:243–270, 2004. 10

[WBP+03] Norman Wilde, Michelle Buckellew, Henry Page, Vaclav Rajlich, and LaTreva Pounds. A comparison of methods for locating features in legacy software. *Journal of Systems and Software*, 65(2):105–114, 2003. 148

[WC89] Yen-Chuen Wei and Chung-Kuan Cheng. Towards efficient hierarchical designs by ratio cut partitioning. *Proceedings of the IEEE International Conference on Computer-Aided Design (ICCAD 1989)*, pages 298–301, 1989. 8, 53, 61

[WC91] Yen-Chuen Wei and Chung-Kuan Cheng. Ratio cut partitioning for hierarchical design. *IEEE Transactions on Computer-Aided Design*, 10(7):911–921, 1991. 53, 61

[Wei81] Mark Weiser. Program slicing. In *Proceedings of the 5th International Conference on Software Engineering (ICSE 1981)*, pages 439–449. IEEE Computer Society, 1981. 147

[Wei82] Mark Weiser. Programmers use slices when debugging. *Communications of the ACM*, 25(7):446–452, 1982. 148

[Wen05] Norman Wenzel. Dokumentation und Bewertung der Abhängigkeiten in großen Software-Systemen (in German). Bachelor thesis, University of Technology at Cottbus, 2005. 240, 247

[WF94] Stanley Wasserman and Katherine Faust. *Social Network Analysis: Methods and Applications*. Cambridge University Press, Cambridge, UK, 1994. 10, 11, 78, 128

[WH92] Norman Wilde and Ross Huitt. Maintenance support for object-oriented programs. *IEEE Transactions on Software Engineering*, 18(12):1038–1044, 1992. 147

[WL86] Mark Weiser and Jim Lyle. Experiments on slicing-based debugging aids. In E. Soloway and S. Iyengar, editors, *Empirical Studies of Programmers*, pages 187–197, Norwood, NJ, 1986. Ablex Publishing. 148

[WS01] Song Wang and Jeffrey Mark Siskind. Image segmentation with minimum mean cut. In *Proceedings of the 8th International Conference on Computer Vision (ICCV 2001)*, pages 517–524. IEEE Computer Society, 2001. 53

[WS03] Song Wang and Jeffrey Mark Siskind. Image segmentation with ratio cut. *IEEE Transaction on Pattern Analysis and Machine Intelligence*, 25(6):675–690, 2003. 53

[XPM06] Xinrong Xie, Denys Poshyvanyk, and Andrian Marcus. Visualization of CVS repository information. In *Proceedings of the 13th Working Conference on Reverse Engineering (WCRE 2006)*, pages 231–242. IEEE Computer Society, 2006. 154

[XQZ+05] Baowen Xu, Ju Qian, Xiaofang Zhang, Zhongqiang Wu, and Lin Chen. A brief survey of program slicing. *ACM SIGSOFT Software Engineering Notes*, 30(2):1–36, 2005. 147

[XS06] Zhenchang Xing and Eleni Stroulia. Understanding the evolution and co-evolution of classes in object-oriented systems. *International Journal of Software Engineering and Knowledge Engineering*, 16(1):23–52, 2006. 154

[XW05] Rui Xu and Donald Wunsch II. Survey of clustering algorithms. *IEEE Transactions on Neural Networks*, 16(3):645–678, 2005. 49, 106

[YCL92] Ching-Wei Yeh, Chung-Kuan Cheng, and Ting-Ting Y. Lin. A probabilistic multicommodity-flow solution to circuit clustering problems. *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design (ICCAD 1992)*, pages 428–431, 1992. 54

[YCL95] Ching-Wei Yeh, Chung-Kuan Cheng, and Ting-Ting Y. Lin. Circuit clustering using a stochastic flow injection method. *IEEE Transactions on Computer-Aided Design*, 14(2):154–162, 1995. 54

[YCM78] Stephen S. Yau, James S. Collofello, and T. MacGregor. Ripple effect analysis of software maintenance. In *Proceedings of the Computer Software and Applications Conference (COMPSAC 1978)*, pages 60–65. IEEE Computer Society, 1978. 148

[YMNCC04] Annie T. T. Ying, Gail C. Murphy, Raymond T. Ng, and Mark Chu-Carroll. Predicting source code changes by mining change history. *IEEE Transactions on Software Engineering*, 30(9):574–586, 2004. 154, 156, 176, 235

[YS03] Stella X. Yu and Jianbo Shi. Multiclass spectral clustering. In *Proceedings of the 9th International Conference on Computer Vision (ICCV 2003)*, pages 313–319. IEEE Computer Society, 2003. 57, 58

[Zac77] Wayne W. Zachary. An information flow model for conflict and fission in small groups. *Journal of Anthropological Research*, 33:452–473, 1977. 231

[ZDZ03] Thomas Zimmermann, Stephan Diehl, and Andreas Zeller. How history justifies system architecture (or not). In *Proceedings of the 6th International Workshop on Principles of Software Evolution (IWPSE 2003)*, pages 73–83. IEEE Computer Society, 2003. 154, 156, 160, 176, 177, 235

[ZGH07] Lijie Zou, Michael W. Godfrey, and Ahmed E. Hassan. Detecting interaction coupling from task interaction histories. In *Proceedings of the 15th International Conference on Program Comprehension (ICPC 2007)*, pages 135–144. IEEE Computer Society, 2007. 236

[ZK04] Ying Zhao and George Karypis. Empirical and theoretical comparisons of selected criterion functions for document clustering. *Machine Learning*, 55(3):311–331, 2004. 61

[ZKWZ06] Thomas Zimmermann, Sunghun Kim, E. James Whitehead Jr., and Andreas Zeller. Mining version archives for co-changed lines. In *Proceedings of the 3rd International Workshop on Mining Software Repositories (MSR 2006)*, pages 72–75, 2006. 236

[Zus98] Horst Zuse. *A Framework of Software Measurement*. Walter de Gruyter, Berlin, 1998. 10, 12

[ZW04] Thomas Zimmermann and Peter Weißgerber. Preprocessing CVS data for fine-grained analysis. In *Proceedings of the 1st International Workshop on Mining Software Repositories (MSR 2004)*, pages 2–6, 2004. 235, 236, 237

[ZWDZ05] Thomas Zimmermann, Peter Weißgerber, Stephan Diehl, and Andreas Zeller. Mining version histories to guide software changes. *IEEE Transactions on Software Engineering*, 31(6):429–445, 2005. 156, 176, 177, 235

[ZZL+06] Wei Zhao, Lu Zhang, Yin Liu, Jiasu Sun, and Fuqing Yang. SNIAFL: Towards a static non-interactive approach to feature location. *ACM Transactions on Software Engineering and Methodology*, 15(2):195–226, 2006. 149

Index