Heuristics for Laying Out Information Graphs

Lenwood Heath and Joseph W. Lavinus

TR 93-27

Department of Computer Science
Virginia Polytechnic Institute and State University
Blacksburg, Virginia 24061

August 27, 1993

Submitted to COMPUTING, 1993.

Heuristics for Laying Out Information Graphs[*]

Lenwood S. Heath
Department of Computer Science
Virginia Polytechnic Institute and State University
Blacksburg, Virginia 24061-0106, USA
FAX: (703) 231–6075

Joseph W. Lavinus
Department of Computer Science
University of Virginia
Charlottesville, Virginia 22903, USA
FAX: (804) 982–2114

Running Title:
Laying Out Information Graphs

[*]The authors gratefully acknowledge the support of National Science Foundation grant IRI–9116991.

Abstract — Zusammenfassung

Heuristics for Laying Out Information Graphs. The concept of an information graph is introduced as a representation for object-oriented databases. The retrieval layout problem is an optimization problem defined over the class of information graphs. The layout abstracts the space efficiency of representing the database as well as the time efficiency of information retrieval. Heuristics for the retrieval layout problem are identified and evaluated experimentally. A new heuristic, connectivity traversal, is found to be fast and to produce high quality layouts.
AMS Subject Classifications: 05C99, 68P20, 68R10, 90C27
Key words: Graph layout, graph partitioning, combinatorial optimization, information retrieval.

Heuristiken für das Layout von Informationsgraphen. Der Begriff eines Informationsgraphen wird als eine Repräsentation für objektorientierte Datenbanken vorgestellt. Das Wiederauffinden von Informationen ergibt sich als Optimierungsproblem bezüglich der so entstandener Informationsgraphen. Das Layout stellt nicht nur den Raumbedarf der Datenbank sondern auch den Zeitbedarf des Wiederauffindens der Informationen dar. Heuristiken für das Wiederauffindensproblem werden identifiziert und experimentell ausgewertet. Eine neue Heuristik—connectivity traversal—ergibt sowohl ein schnelles Widerauffindensverfahrens wie auch qualitativ hochwertige Layouts.

1 Introduction

Object-oriented databases are a significant topic of recent study in the fields of database management and information retrieval. A common application of such databases is the storage of many small pieces of information (objects) connected by well-defined relationships. Examples include a dictionary of the English language (with connections for synonyms, antonyms, is-a, and other relationships) and a bibliographic archive (with connections for cited-by, same-author, same-subject, and other relationships). A natural representation for such databases is the *information graph* [15] in which the objects correspond to vertices in the graph and the relationships correspond to edges. In an information graph, each vertex has a *size* (in bytes) sufficient to hold its associated object. Since the media that store information graphs cannot efficiently handle retrieval of individual small objects, the objects are stored in fixed-size *pages*, each sufficient to hold several objects.

An important optimization issue for information graphs is the basis for assigning vertices to pages. This assignment must allow space-efficient storage of the information graph and time-efficient retrieval of its contents. Retrieval in an information graph may follow a number of patterns; in particular, retrieval of small neighborhoods within the graph must be efficient. These criteria suggest the *retrieval layout problem* for information graphs: find a layout of the vertices (an assignment of vertices to pages) that achieves time-efficient retrieval and space-efficient storage. Space efficiency, achieved through vertex packing, is fundamentally the classic bin packing problem [12]. The added requirement of efficient retrieval makes retrieval layout a significant new problem.

Because direct measurement of the retrieval efficiency of a layout is expensive on graphs of even moderate size, a number of easily evaluated objective functions are investigated as approximations of retrieval efficiency. Among these objective functions are total edge length, bandwidth, and cutwidth; optimizing over these three objective functions is NP-complete. We also introduce two new objective functions, *number of interpage edges* and *number of long edges*; optimizing over these objective functions is also NP-complete [14]. As a consequence, we turn to heuristics for optimizing these objective functions. The heuristics are ranked according to their efficiency in optimizing the objective functions by testing them with various input graphs, including actual databases, random graphs, geometric graphs, and two graph models designed specifically for the retrieval layout problem: *gravitational graphs* and *quaesitum graphs.* The actual databases tested are a medical thesaurus called MeSH and a bibliographic archive from *Communications of the ACM.* Among the heuristics we consider are a greedy heuristic, breadth-first and depth-first traversal, simulated annealing, tabu search, two variants of genetic algorithms, and recursive decomposition. In addition, we test a *hierarchical hybrid* approach consisting of recursive decomposition combined with more expensive heuristics such as simulated annealing and tabu search. Finally, we introduce *connectivity traversal,* a fast heuristic tailored to the retrieval layout problem.

Our intention is to identify those objective functions that are clearly useful in finding good layouts. Consequently, we rank each objective function according to its value as a measure of retrieval efficiency. This ranking is achieved by means of a simulation of various retrieval patterns in the information graph, resulting in an approximation of the cost of retrieval. These retrieval patterns model actual retrieval operations that might be performed by a user or by a database application. The patterns studied are *random browse*, *breadth-first retrieval*, and *depth-first retrieval.* By comparing the retrieval cost of layouts produced by optimizing different objective functions, we rank the objective functions according to their value as approximations of retrieval cost.

Chen [3] describes an object-oriented database system called the Large External object-oriented Network Database (LEND). LEND supports the storage and retrieval of information graphs. To enhance LEND performance, Chen informally examines the retrieval layout problem. However, he tests only one layout heuristic (recursive decomposition) and only examines that heuristic with respect to one objective function (number of interpage edges) and one type of retrieval pattern (random browse). Our study pursues the problem to a greater depth than previous research.

Information graphs for information retrieval applications are quite large—hundreds of thousands of vertices and tens of megabytes in size—so heuristics of low time complexity (nearly linear) are essential. Previous experimentation with graph layout heuristics has been limited to graphs with 1,000 or fewer vertices (for examples, see [14]). The graphs we consider are fairly sparse like real information graphs, that is, they have low average degree. In any case, moderately dense random graphs cannot be laid out well (Theorem 1). We also assume that an information graph is *static*; since its ultimate home is probably a high-capacity medium such as CD-ROM, layout is performed infrequently, and the database remains unchanged for long periods of time. Thus, it is cost-effective to spend a great deal of computing time—hours or even days—to achieve a high-quality layout.

The remainder of the paper is organized as follows. Section 2 defines some notation for the retrieval layout problem and describes the objective functions. Section 3 describes the heuristics tested, and Section 4 describes the types of graphs used as input to these heuristics. Section 5 describes the framework used for correlating the objective functions with retrieval cost, the experiments performed, and the performance of the heuristics. Finally, Section 6 concludes with a summary and some possible directions for future work.

2 Basics

Notation. An *information graph* is an undirected graph $G = (V, \rho, E)$ where

- V is the set of vertices;
- The function $\rho : V \rightarrow \mathbb{Z}$ gives the *size* of a vertex in bytes; and
- E is the set of edges.

The *size* of a subgraph of G is the sum of the sizes of its vertices. The *neighbor set* of v is $\Gamma(v) = \{u : (u,v) \in E\}$, and the *degree* of a vertex v is $\delta(v) = |\Gamma(v)|$. The *maximum degree* of G is $D(G)$, and the *average degree* of G is $\overline{\delta}(G)$. Each edge is stored in the adjacency list of both of its endpoints; thus, the size of each vertex must be sufficient to store pointers to its neighbors. A

Objective function	Definition
Total edge length	$\mathcal{TL}(\lambda) = \sum_{(u,v)\in E} \Delta_\lambda(u,v)$
Number of interpage edges	$\mathcal{NI}(\lambda) = \lvert\{(u,v) \in E : \lambda(u) \neq \lambda(v)\}\rvert$
Bandwidth	$\mathcal{BW}(\lambda) = \max_{(u,v)\in E} \Delta_\lambda(u,v)$
Number of long edges	$\mathcal{NL}(\lambda, L) = \lvert\{(u,v) \in E : \Delta_\lambda(u,v) > L\}\rvert$
Cutwidth	$\mathcal{CW}(\lambda) = \max_{i\in\{2,\ldots,R\}} \lvert\{(u,v) \in E : \lambda(u) < i \leq \lambda(v)\}\rvert$

Table 1: Objective functions for the retrieval layout problem.

constant P defines the page size in bytes. A *layout* λ is a function that maps vertices to pages such that the sum of the sizes of the vertices in any page does not exceed P. A *partial layout* is a layout function whose domain is restricted to a subset of V. Since most layout functions are many-to-one, a layout function rarely maps to the full range 1 to $|V|$. For a layout λ, it is assumed that the layout begins on page 1, and the *page utilization* $R(\lambda)$ is the highest page index used by λ. In a layout λ, the *length* of an edge (u, v) is the linear distance in pages from u to v; stated mathematically: $\Delta_\lambda(u,v) = |\lambda(v) - \lambda(u)|$.

Objective functions. An objective function for the retrieval layout problem associates an *objective value* with each layout. An objective function must be easy to evaluate, and it must be possible to compute good lower bounds on the objective function for partial layouts created during construction of a complete layout. Obviously, an objective function must also be related to the quality of a layout produced by optimizing it. The candidate objective functions we examine are defined in Table 1. Optimizing over any of these is NP-complete; Lavinus [14] provides references and proofs of NP-completeness. For each objective function, a lower objective value indicates a higher-quality layout, so the related problem is always a minimization problem.

3 Heuristics

The NP-completeness of optimizing over the objective functions suggests that there is no efficient algorithm to solve the retrieval layout problem exactly. Thus, we examine a number of heuristics for

Heuristic	Time Complexity	Reference
Random	$O(\|V\|)$	[14]
Greedy	$O(\bar{\delta}\|V\|^2)$	[14]
BFT/DFT	$O(\|V\| + \|E\|)$	—
Connectivity traversal	$O(\|V\| + \|E\|)$	[14]
Clustering traversal	$O(\|V\|)$	[16]
Recursive decomposition	$O(\|E\| \log \|V\|)$	[3]
Local optimization	unpredictable	[13]
Simulated annealing	unpredictable	[18]
Tabu search	$O(m(R(\lambda) \cdot \|E\| + t))$	[8]
Genetic algorithms	unpredictable	[9]

Table 2: Summary of heuristics examined.

producing good solutions. These heuristics are described briefly in this section and are summarized in Table 2. Full descriptions and implementation details appear in Lavinus [14].

Random solution. Due to the exponential number of layouts, a random solution is very likely to be a poor one; nonetheless, we implement it for comparison with better heuristics.

Greedy heuristic. The greedy heuristic inserts vertices into a partial layout from page 1 to page $R(\lambda)$ such that at each step, the objective value of the current partial layout is minimized.

Breadth-first and depth-first traversal. The vertices are assigned to pages from page 1 to page $R(\lambda)$ in the order specified by a breadth-first or depth-first search of the graph.

Connectivity traversal. This heuristic, devised specifically for the retrieval layout problem, also relies on a traversal of the graph, but the traversal strategy is more complex. In connectivity traversal, an initial vertex is chosen uniformly at random, and subsequent vertices are inserted into the layout from page 1 to page $R(\lambda)$; at each step, the vertex chosen is the one that is most heavily connected to those vertices already inserted in the current page. While one may contrive an artificial example graph for which connectivity traversal performs quite poorly [14], such graphs are unlikely to occur in practice.

Clustering traversal. In clustering traversal [16], vertices are inserted from page 1 to page $R(\lambda)$, and the next vertex chosen is always an arbitrary vertex of minimum degree. There are graphs for which clustering traversal performs very badly [14], and unlike the bad examples for connectivity traversal, such graphs are likely to occur in practice.

Recursive decomposition. A recursive decomposition heuristic partitions V into two subsets, and then recursively partitions the subgraphs induced by these subsets in the same manner, until some threshold is reached. Our implementation uses the Fiduccia-Mattheyses heuristic [5] to partition each subgraph into two smaller subgraphs such that the number of edges with endpoints in different subgraphs is minimized and such that the sizes of the two subgraphs are roughly equal. The decomposition stops when the subsets each fit in a single page. We also test a faster heuristic called greedy partitioning. This heuristic, reminiscent of connectivity traversal, begins by placing a different randomly selected vertex in each of the two partitions, then inserts each subsequent vertex into the partition to which it is most heavily connected.

Local optimization. Local optimization is a simple iterative improvement heuristic; from an initial (usually random) layout, it locally modifies the layout if the modification improves the objective value. In our implementation, this modification (called a *move*) takes a vertex from its current page and places it in a different page, provided there is space in the new page for the vertex. Some objective functions—$\mathcal{NI}$, $\mathcal{TL}$, and $\mathcal{NL}$—can be computed incrementally, and thus each move can be made in effectively constant time. For the others—$\mathcal{BW}$ and $\mathcal{CW}$—the objective function must be entirely recomputed after each move.

Simulated annealing. Simulated annealing [18] is a well-known iterative improvement heuristic designed to avoid entrapment in globally poor local optima. It accomplishes this aim by incorporating the notion of *temperature*. As in local optimization, improving moves are always taken. However, moves that increase the objective value of the layout may also be taken with probability directly dependent on the temperature and inversely dependent on the change in objective value. As the temperature is slowly decreased, the solution converges to some locally optimal value, which is typically a much better solution than one produced by local optimization. Our implementation is similar to that of Johnson, Aragon, McGeoch, and Schevon [13].

Tabu search. Tabu search [8] takes a different approach to iterative improvement: rather than avoiding local optima, it provides a mechanism for escaping local optima after they are encountered. The key notions in tabu search are *aggressiveness* and a *tabu list* of forbidden moves. Aggressiveness means that rather than examining randomly chosen moves, tabu search examines all possible moves at each step and takes the one that results in the best objective value. (Note that this does not imply an improvement in objective value; if the current layout is a local optimum, then tabu search chooses the non-tabu move that increases the objective value by the smallest amount.) The tabu list, which typically contains the last t moves that were made (where t is a small constant), prevents the heuristic from cycling in and out of a local optimum. The heuristic allows a tabu move to override its tabu status if it results in a layout with lower objective value than any seen so far. Tabu search halts when a certain number of moves have elapsed without finding a new best layout, or when the total number of moves exceeds a constant m. Thus, the worst-case time complexity of tabu search is $O(m(R(\lambda) \cdot |E| + t))$ (though in practice this bound is quite pessimistic).

Genetic algorithms. A genetic algorithm [9] maintains a current generation of solutions, which are mated with one another by means of a *crossover* operator such that solutions with lower objective values are more likely to survive to the next generation. We implement genetic algorithms for the retrieval layout problem using two different crossover operators: *partially mapped crossover (PMX)* [4] and *order crossover (OX)* [17].

4 Graphs

This section describes the types of graphs used as input for testing the heuristics. For all randomly generated graphs, vertex size is either constant or drawn from a normal distribution, which is a reasonable approximation to the vertex sizes found in actual databases.

Actual databases. The first, and perhaps most important, type of input is two actual databases. The MeSH (for *Me*dical *S*ubject *H*eadings) database [11] is a medical thesaurus containing 287, 724

Database	$\overline{\delta}$	$\sigma^2(\delta)$	D	$\overline{\rho}$	$\sigma^2(\rho)$	$\max\rho$	$\min\rho$
CACM	21.480	31.199	592	63.362	50.836	270	5
MeSH	2.360	9.407	1800	58.491	93.391	1510	0

Table 3: Vertex degree and vertex size statistics.

vertices and 339,489 edges. CACM is a bibliographic database extracted from *Communications of the ACM* [6] and contains 6,093 vertices and 65,438 edges. Vertex degree and vertex size statistics for MeSH and CACM are shown in Table 3.

Random graphs. Random graphs [2] are also used as input. In the simplest model, $R_{n,q}$ denotes the class of n-vertex random graphs in which each edge is present with probability q. We conjecture that random graphs are not an accurate model of the real problem domain. Moreover, we [10] show that a random graph cannot be laid out well, in a sense embodied in the following result.

Theorem 1 (Heath and Lavinus [10]) *Let G be a random graph from $R_{n,q}$. Suppose that page size is $P = o(n)$, that ε is a positive constant, and that γ is a constant satisfying $1 > \gamma \geq q$. Let E be the expected number of interpage edges in a random layout of G. If the expected degree of G is $\omega(\ln n)$, then, as $n \to \infty$,*

$$\Pr[\mathcal{NI}(G) \leq (1-\varepsilon)E] \to 0.$$

In other words, if the degree of a random graph is slightly greater than logarithmic, then the number of interpage edges in a random layout is asymptotically about same as the number of interpage edges in an optimal layout.

Geometric graphs. Johnson, Aragon, McGeoch, and Schevon [13] discuss a class of randomly generated graphs they call *geometric graphs.* An element of the class $U_{n,d}$ of n-vertex, distance-d geometric graphs is generated as follows. Each vertex is a point chosen uniformly at random in the unit square. An edge exists between two points u and v if $\mathrm{L}_2(u,v) < d$, where L_2 is Euclidean distance. The expected average degree of a geometric graph is $n\pi d^2$.

Gravitational graphs. Geometric graphs are too simple to accurately mirror the complex structure of an actual information graph; thus, we devise a modification called *gravitational graphs.* In a gravitational graph, each vertex v is assigned a random *mass* $m(v)$ drawn from a lognormal distribution (see [1]). For the class $M_{n,d}$ of n-vertex gravitational graphs, an edge exists between two vertices u and v if

$$\frac{\mathrm{L}_2(u,v)}{\max\{m(u),m(v)\}} < d.$$

Thus, vertices with high mass are adjacent to more distant vertices than those with lower mass. the resulting degree distribution resembles that of the MeSH and CACM databases described above. The expected average degree of a gravitational graph is somewhat greater than $n\pi ed^2$.

Quaesitum graphs. The final type of input is a *quaesitum graph*, also devised specifically for the retrieval layout problem, which is a graph generated in such a way that a good upper bound on its optimum objective value is known. An n-vertex quaesitum graph from $Q_{n,p,q}$ is generated as follows. Assign vertices to pages as in a random layout. Then, add an edge between a pair of vertices (u,v) with probability p for intrapage edges and $q/\Delta_\lambda(u,v)$ for interpage edges, where $p \gg q$. The resulting graph is heavily connected within pages and loosely connected between pages, and the probability of the existence of an edge is inversely correlated with its length. Thus, the layout from which the graph is generated is likely to be nearly optimal with respect to $\mathcal{NI}$, $\mathcal{TL}$, and $\mathcal{NL}$ and good with respect to $\mathcal{BW}$ and $\mathcal{CW}$, so the results of applying the heuristics to the graph can be compared against the objective values of this original layout.

5 Experiments

We use empirical testing to analyze three aspects of heuristics and objective functions for the information retrieval problem:

1. Time complexity of each heuristic;

2. Solution quality of each heuristic; and

3. Correlation between an objection function and actual retrieval cost.

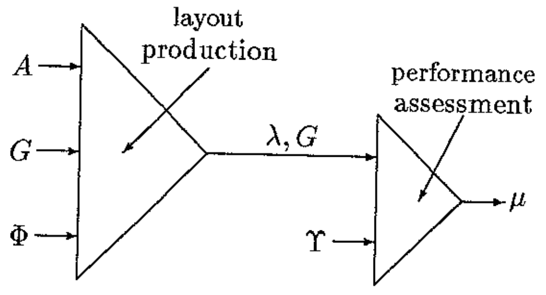


Figure 1: Flowchart of the testing process. A is a layout heuristic, Υ is a retrieval sequence, and μ is a measure of retrieval cost.

To study these three aspects, we apply the layout heuristics to various inputs, using different objective functions and combinations thereof. In addition, some heuristics (such as simulated annealing) depend upon parameters for which good values cannot be determined analytically. Thus, good settings for these parameters are determined experimentally (as in [13]).

A suite of experiments exercises each heuristic with respect to various objective functions and tests the results with respect to different retrieval patterns in the resulting layouts. The diagram shown in Figure 1 illustrates the testing process used. The performance assessment and layout production phases of the process are covered in the subsequent sections.

5.1 Performance assessment

The phase of this study labeled *performance assessment* in Figure 1 tests the quality of each objective function (generically denoted Φ) as an approximation of retrieval efficiency. The performance assessment phase involves retrieving vertices from a layout λ according to some retrieval sequence Υ, determining the retrieval cost μ of this traversal, and determining any correlation between $\Phi(\lambda)$ and μ for each Φ.

To approximate retrieval cost in an information graph, we define an access model to approximate access to an information graph by a database application or a human user. The order in which the vertices in the graph are retrieved is defined by a *retrieval sequence* $\Upsilon =< v_1, v_2, \ldots >$ that consists of vertices of G, that may contain repetitions, and that may be infinite. We assume that a *buffer* β

that holds $|\beta|$ pages is maintained in main memory. Let v_f be the vertex most recently read from secondary storage (this is always a vertex in the page most recently read into a page of β). Initially set β to be empty and set v_f to be some vertex in page 1. The *cost* $\mu(v_i)$ of retrieving a vertex v_i is:

$$\mu(v_i) = \begin{cases} \Delta_\lambda(v_f, v_i) & \lambda(v_i) \notin \beta; \\ 0 & \lambda(v_i) \in \beta. \end{cases}$$

The first case corresponds to a *page fault* when $\lambda(v_i)$ replaces a page p in β after the storage hardware seeks a distance of $\Delta_\lambda(v_f, v_i)$ and reads page $\lambda(v_i)$. A *page replacement policy* specifies the choice of p; in our model, p is the least recently used page. The costs of determining whether v_i is in β and choosing p are assumed negligible compared to the cost of accessing a page not in β. In testing a retrieval sequence Υ, the buffer size $|\beta|$ is fixed, and the first c vertices in Υ are retrieved in order. The *retrieval cost* of Υ is then

$$\mu(\Upsilon) = \sum_{i=1}^{c} \mu(v_i).$$

Let $f(\Upsilon)$ be the number of page faults. With respect to media access, f measures the number of times the media is accessed, while μ measures the total seek distance traveled during those accesses. The relative importance of these two measures is dependent on the storage hardware.

In an actual application, $|\beta|$ can be no larger than the available main memory of the machine. In practice, the application is likely to run in a multiuser environment, and $|\beta|$ is much smaller. In LEND applications, where $P = 8192$, a reasonable value is $|\beta| = 20$, which is the value we use in our testing.

A *retrieval strategy* is a procedure that generates a retrieval sequence Υ. The aim of such a strategy is to approximate a retrieval sequence that might result from a user or application using the database. The following retrieval strategies are defined:

- *Random browse.* The initial vertex v_1 in the retrieval sequence is chosen uniformly at random, and each subsequent vertex v_i is chosen uniformly at random from $\Gamma(v_{i-1})$.
- *Breadth-first retrieval.* The initial vertex is chosen uniformly at random, and successive vertices are generated by a breadth-first traversal of the information graph.

- *Depth-first retrieval.* The initial vertex is chosen uniformly at random, and successive vertices are generated by a depth-first traversal of the information graph.

Note that a random browse retrieval sequence is infinite, while a retrieval sequence of the latter two types has length bounded by $|V|$. Random browse models a user who is simply browsing in a hypertext-like system, while the latter two model a systematic search by a database application for some kind of information, as in a relational query.

We first examine the behaviors of these retrieval strategies on a layout of the CACM graph produced by connectivity traversal; this layout has low values for all objective functions. We find that random browse is a low-cost retrieval strategy. This is because it chooses each edge to traverse at random, so it often doubles back on itself, traversing portions of the graph that have already been visited and are likely to be in β. In contrast, breadth-first and depth-first retrieval never revisit an edge and result in higher retrieval cost. We optimize for breadth-first or depth-first retrieval for two reasons. First, in an actual database, these operations are far more time-critical than random browse, since they typically involve a far wider search and are directed by a database application that is much faster than a browsing user. Second, the performance of breadth-first and depth-first retrieval is more statistically stable than random browse.

The purpose of the performance assessment phase is to determine the accuracy of the various objective functions as approximations of retrieval efficiency. Note first that the objective functions are highly codependent. This makes intuitive sense: if many edges have length 0 (i.e., the layout has low $\mathcal{NI}$), then $\mathcal{TL}$ is likely to be low as well; reducing one objective function typically reduces others as a side effect. This codependence makes it difficult to construct experiments comparing the objective functions. However, we are able to contrive layouts that have unnaturally independent objective values for the different objective functions for the purpose of testing the objective functions against each other.

The results of these tests largely reflect intuition. The three objective functions $\mathcal{NL}$, $\mathcal{BW}$, and $\mathcal{CW}$ are almost completely irrelevant; testing on layouts in which the values of $\mathcal{NI}$ and $\mathcal{TL}$ are the same, but in which the values of $\mathcal{NL}$, $\mathcal{BW}$, and $\mathcal{CW}$ differ, demonstrates that their values have no significant correlation with retrieval performance. In fact, for small $|\beta|$, retrieval cost is actually slightly worse for layouts with low $\mathcal{NL}$, $\mathcal{BW}$, and $\mathcal{CW}$; while optimizing these objective functions reduces the number of very long edges, it tends to increase the number of medium-length edges in the layout. This trend reverses for large $|\beta|$, since the endpoints of the medium-length edges are often already in β. The effect of these three objective functions on f is almost nil. It is fortunate that $\mathcal{BW}$ and $\mathcal{CW}$ are not useful objective functions, as their computation is expensive.

As intuition indicates, $\mathcal{NI}$ and $\mathcal{TL}$ are the most relevant objective functions. The testing reported below is designed to determine the relative importance of these two objective functions and to develop some insight into how each reflects upon retrieval performance.

Toward this aim, we devise four layouts of the CACM graph, corresponding to the four combinations of low and high values of $\mathcal{NI}$ and $\mathcal{TL}$. The layout with low values for both objective functions is generated by a run of connectivity traversal, and the layout with both high values is generated by a run of the greedy heuristic. The layout with low $\mathcal{NI}$ and high $\mathcal{TL}$ is generated by randomly shuffling the pages in the layout produced by connectivity traversal. This shuffle leaves $\mathcal{NI}$ unchanged, but deteriorates $\mathcal{TL}$. Producing a layout with high $\mathcal{NI}$ and low $\mathcal{TL}$ is more difficult. We devise such a layout by running simulated annealing with the objective function $\Phi(\lambda) = \mathcal{TL}(\lambda) - \mathcal{NI}(\lambda)$, which serves to minimize $\mathcal{TL}$ while maximizing $\mathcal{NI}$.

Applying the retrieval simulation to these layouts, we find that the influence of $\mathcal{NI}$ and $\mathcal{TL}$ on retrieval performance depends on $|\beta|$. This reflects the fact that $\mathcal{NI}$ measures intrapage locality, whereas $\mathcal{TL}$ measures interpage locality. For $|\beta| = 1$, the cost μ is largely dependent on the probability that the next vertex retrieved is in the current page, which correlates with $\mathcal{NI}$. As $|\beta|$ grows, μ depends less on the probability that the next vertex is in the current page, and more on the probability that it is in a recently accessed page. This probability correlates with $\mathcal{TL}$. However, the layout that optimizes both objective functions outperforms the layout that optimizes either separately, for all values of $|\beta|$. Predictably, number of page faults depends only on $\mathcal{NI}$. Thus, it is desirable to minimize both $\mathcal{NI}$ and $\mathcal{TL}$. The relative weights assigned to these objective functions depend on the buffer size and on the relative costs of seeking (proportional to μ) versus the other overhead involved in a page fault, such as the time required to actually read the page into memory. These costs vary between media and between specific pieces of hardware. For a CD-ROM, seeking is more expensive than reading (see [7], Chapter 6), and thus a higher weight should be assigned to $\mathcal{TL}$. For a magnetic medium such as a hard drive, the reverse is true, and a higher weight should be assigned to $\mathcal{NI}$. In any case, the choice of weights is probably not crucial, and the two objective functions are typically in agreement.

5.2 Layout production

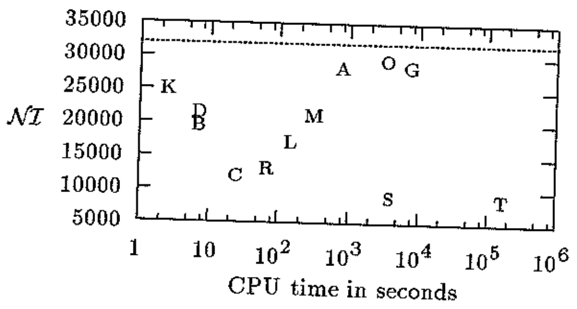
The phase of this study labeled *layout production* in Figure 1 tests the quality of the layouts produced by each heuristic with respect to the various objective functions, as well as its running time. Each test inputs a heuristic A, a graph G, and, for those heuristics that evaluate an objective function (such as the greedy and iterative improvement heuristics), an objective function Φ. The output is a layout λ. A heuristic A *dominates* (empirically) a heuristic B if it is both faster and produces a layout with a better objective value.

First-cut testing. Since the number of possible combinations of heuristics and input graphs is large, three representative graphs are tested on all heuristics to make a first cut, i.e., to eliminate heuristics that are dominated by some other heuristic. The three graphs are the CACM database, a geometric graph from $U_{10000,0.01}$, and a gravitational graph from $M_{10000,0.01}$.

Figures 2 and 3 plot the performance of each heuristic versus its running time on the three first-cut graphs. The heuristics are represented by the single-letter mnemonics listed in Table 4. Each heuristic is run at least 10 times, and beyond that, a sufficient number of times to reduce the variances in objective value and running time to less than 1% of their mean. The dashed horizontal line near the top of each plot indicates the value of $\mathcal{NI}$ for a random layout. The testing platform is a DECstation$^{\text{TM}}$ 5000/200 with 40Mb of main memory. In all three tests, connectivity traversal is clearly the best heuristic, in the sense that it always gives a high-quality solution more quickly than other heuristics that give quality solutions. We eliminate from further consideration those heuristics that are dominated by connectivity traversal in all three tests; these are genetic algorithms (G and O), the clustering heuristic (M), and the greedy heuristic (A). Connectivity traversal also dominates recursive decomposition for the case where decomposition progresses until the subgraphs fit in a single page. Thus, recursive decomposition is tested further only in the context of the hierarchical hybrid approach (described later).

Heuristic	Mnemonic	Heuristic	Mnemonic
Greedy heuristic	A	Greedy recursive decomposition	K
Breadth-first traversal	B	Local optimization	L
Depth-first traversal	D	Simulated annealing	S
Connectivity traversal	C	Tabu search	T
Clustering	M	Genetic algorithm (PMX)	G
FM recursive decomposition	R	Genetic algorithm (OX)	O

Table 4: The mnemonics used for each heuristic.



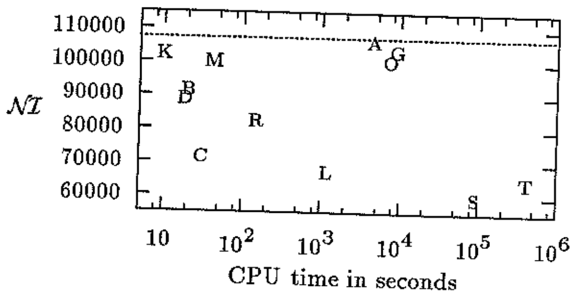
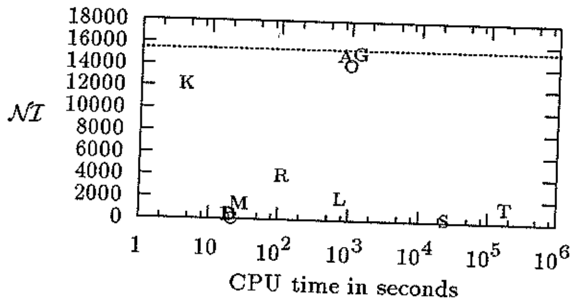


Figure 2: $\mathcal{NI}$ versus running time for all heuristics on CACM (left) and $M_{10000,0.01}$ (right).



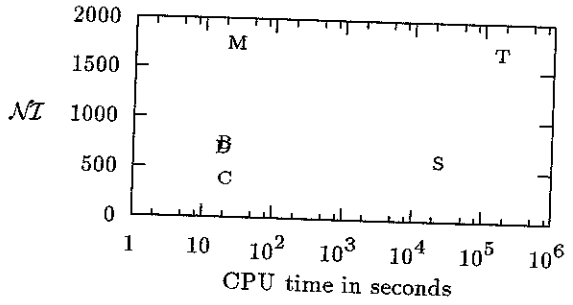


Figure 3: $\mathcal{NI}$ versus running time for all heuristics on $U_{10000,0.01}$ (left), and enlargement of the lower portion thereof (right).

Connectivity traversal outperforms several of the simpler heuristics in layout quality but not in running time. These are BFT, DFT, and greedy recursive decomposition. For the sake of comprehensiveness, these are tested further.

Local optimization performs rather sporadically; its solution quality is mediocre for both CACM and $U_{10000,0.01}$, but it attains a good solution for $M_{10000,0.01}$. In all cases, local optimization requires significantly more time than connectivity traversal. In addition, local optimization has an advantage that the other iterative improvement heuristics lack: its running time is strongly correlated with the amount of layout improvement it achieves.

Simulated annealing performs as expected: it produces high-quality layouts but requires large amounts of computing time. Its running time is impractical for large graphs, but it is examined further in the context of the hierarchical hybrid approach.

Tabu search also requires large amounts of computing time, and in two of the three first-cut tests, produces results no better than connectivity traversal. It is examined further as part of the hierarchical hybrid approach but is too slow to be of practical use on large information graphs.

Further testing. While recursive decomposition performs rather poorly if the decomposition progresses until each subgraph fits in a single page, an approach that warrants further investigation is *hierarchical hybrid*, i.e., recursive decomposition into subgraphs that are small enough to apply more expensive heuristics to them. Specifically, the approach taken here is to run recursive decomposition until each subgraph has size less than or equal to kP, where k is a small constant. Simulated annealing, tabu search, or connectivity traversal is then run on each resulting subgraph, and the results appended to form a complete layout. The results indicate that hierarchical hybrid using simulated annealing or tabu search is a viable approach that sometimes outperforms connectivity traversal, though it requires far more running time.

Table 5 shows the objective values and running times of BFT, DFT, greedy recursive decomposition, connectivity traversal, and hierarchical hybrid using simulated annealing (S_4) and tabu search (T_4) on subgraphs of size $4P$. The inputs are MeSH and three 100,000-vertex graphs: a random graph, a geometric graph, and a gravitational graph. The results for these large graphs emphasize the need for linear-time heuristics; even simple linear-time heuristics such as BFT require approximately a day of computer time to lay out MeSH. We suspect that a contributing

A	$\mathcal{R}$	$\mathcal{NI}$	$\mathcal{TL}$	CPU time
		MeSH		
B	2082	191361	92371362	24h31m
D	2082	189109	137024787	25h18m
C	2252	117603	70609237	40h0m
K	4024	165315	233144072	8h34m
S_4	3632	108321	102736125	89h38m
T_4	3864	134223	178361641	93h16m
		$M_{100000,0.001}$		
B	1244	79691	8126969	38m
D	1244	54618	8512165	34m
C	1240	47761	8259206	39m
K	2003	72371	10371612	29m
S_4	1562	43259	10274620	16h42m
T_4	1741	48231	12235712	21h17m

A	$\mathcal{R}$	$\mathcal{NI}$	$\mathcal{TL}$	CPU time
		$U_{100000,0.003}$		
B	1244	7984	7984	33m
D	1244	6718	8585	33m
C	1241	4424	4434	35m
K	1992	8121	10278	21m
S_4	1649	23381	50273	19h37m
T_4	1672	25235	72358	24h11m
		$R_{100000,0.0001}$		
B	1242	213428	794223	48m
D	1242	203139	1037233	47m
C	1239	181342	642312	1h11m
K	2107	199324	2381741	31m
S_4	1847	177324	1542323	18h13m
T_4	1889	186331	2033851	21h47m

Table 5: Results of further testing on four large graphs.

factor to these large running times is virtual memory. Most of the heuristics examined exhibit poor locality of reference, and for graphs larger than the available main memory, a great deal of time is spent swapping pages to and from secondary storage. A notable exception is recursive decomposition, which has high reference locality, since it spends the majority of its execution working on small subgraphs. In Table 5, note the difference between the running times of greedy recursive decomposition and those of connectivity traversal, BFT, or DFT. In spite of the fact that recursive decomposition has $O(n \log n)$ time complexity while the other three are linear, greedy recursive decomposition is significantly faster. An improvement in running time might be realized by improving the reference locality of the better heuristics, or by devising an explicitly disk-based approach in which the heuristic handles its own paging rather than relying on the operating system.

6 Conclusions

This is the first formal study of the retrieval layout problem, an optimization problem for information graphs. Using a simulation of various retrieval patterns in an information graph, we have shown that among a variety of objective functions, number of interpage edges and total edge length are sufficient to approximate retrieval performance in an information graph. We have introduced a linear-time heuristic called connectivity traversal, and by comparing its performance with that of many other heuristics, we have shown that it efficiently and consistently produces good layouts of several types of information graphs. Hierarchical hybrid with simulated annealing sometimes produces better layouts than connectivity traversal, but its performance is erratic and it occasionally produces layouts far worse than connectivity traversal. In addition, its running time is significantly higher than that of connectivity traversal. All the other heuristics either produce consistently poorer results than these or require an impractical amount of computing time.

We are confident that further empirical study and deeper theoretical insights will lead to improved heuristics for the retrieval layout problem. Our study indicates that connectivity traversal is a fruitful starting point for future research.

References

[1] L. J. Bain and M. Engelhardt, *Introduction to Probability and Mathematical Statistics*, Duxbury Press, Belmont, 1992.

[2] B. Bollobás, *Random Graphs*, Academic Press, Orlando, Florida, 1985.

[3] Q. F. Chen, *An Object-Oriented Database System for Efficient Information Retrieval Applications*, PhD thesis, Department of Computer Science, Virginia Polytechnic Institute and State University, 1992.

[4] P. J. Denning, *Genetic algorithms*, American Scientist, 80 (1992), pp. 12–14.

[5] C. M. Fiduccia and R. M. Mattheyses, *A linear-time heuristic for improving network partitions*, in Proceedings of the 19th IEEE Design Automation Conference, 1982, pp. 175–181.

[6] E. A. Fox, *Characterization of two new experimental collections in computer and information science containing textual and bibliographic concepts*, Tech. Rep. 83–561, Cornell University, 1983.

[7] W. B. Frakes and R. Baeza-Yates, eds., *Information Retrieval: Data Structures and Algorithms*, Prentice Hall, Englewood Cliffs, 1992.

[8] F. Glover, *Tabu search—part I*, ORSA Journal on Computing, 1 (1989), pp. 190–206.

[9] D. E. Goldberg, *Genetic Algorithms in Search, Optimization, and Machine Learning*, Addison-Wesley, Reading, Massachusetts, 1989.

[10] L. S. Heath and J. W. Lavinus, *Optimal and random partitions of random graphs*. Submitted; available as Technical Report TR 93-24, Department of Computer Science, Virginia Tech, 1993.

[11] S. M. Humphrey and N. E. Miller, *The NLM indexing aid project*, in Proceedings of the 49th Annual Meeting of the American Society for Information Science, 1986, pp. 106–112.

[12] D. S. Johnson, *Fast algorithms for bin packing*, Journal of Computer and System Sciences, 8 (1974), pp. 272–314.

[13] D. S. Johnson, C. R. Aragon, L. A. McGeoch, and C. Schevon, *Optimization by simulated annealing; part I, graph partitioning*, Operations Research, 37 (1989), pp. 865–892.

[14] J. W. Lavinus, *Heuristics for laying out information graphs*, Master's thesis, Department of Computer Science, Virginia Polytechnic Institute and State University, 1992. Available as Technical Report ST92–01.

[15] H. V. D. Parunak, *Ordering the information graph*, in Hypertext/Hypermedia Handbook, E. Berk and J. Devlin, eds., McGraw-Hill, 1991.

[16] A. L. Sangiovanni-Vincentelli, L. Chen, and L. O. Chua, *An efficient heuristic cluster algorithm for tearing large-scale networks*, IEEE Transactions on Circuits and Systems, CAS–24 (1977), pp. 709–717.

[17] K. Shahookar and P. Mazumder, *A genetic approach to standard cell placement using meta-genetic parameter optimization*, IEEE Transactions on Computer-Aided Design, 9 (1990), pp. 500–511.

[18] P. J. M. van Laarhoven and E. H. L. Aarts, *Simulated Annealing: Theory and Applications*, D. Reidel, Boston, 1987.