View metadata, citation and similar papers at core.ac.uk
brought to you by CORE
provided by KITopen

A Scalability Study of Evolutionary Algorithms for Clustering

von

cand.inform.
Stefan Rolf Bach

DIPLOMARBEIT

Institut für Angewandte Informatik
und Formale Beschreibungsverfahren
Universität Karlsruhe

Referent: Prof. Dr. Hartmut Schmeck
Betreuer: Assoc.Prof. Dr. Jürgen Branke University of Warwick, UK
Asst.Prof. Dr. A. Şima Uyar Istanbul Technical University, Turkey

Karlsruhe, 13.05.2009

KIT - The Cooperation between the Forschungszentrum
Karlsruhe GmbH and the Universität Karlsruhe (TH)

Forschungszentrum Karlsruhe
in der Helmholtz-Gemeinschaft

Universität Karlsruhe (TH)
Research University · founded 1825

Stefan R. Bach. A Scalability Study of Evolutionary Algorithms for Clustering. Diploma thesis, University of Karlsruhe, 2009. Available from: `http://digbib.ubka.uni-karlsruhe.de/volltexte/1000011444`

Hiermit erkläre ich, die vorliegende Arbeit selbständig verfasst und keine anderen als die angegebenen Literaturhilfsmittel verwendet zu haben.

I hereby declare that this thesis is a work of my own, and that only cited sources have been used.

Karlsruhe, 13. Mai 2009

Stefan Bach

Acknowledgments

I have worked on the main part of this thesis during my time at the İstanbul Teknik Üniversitesi (ITÜ) in Istanbul, Turkey. This would have never been possible, without the great support of my truly cosmopolitan advisors, Dr. Jürgen Branke and Dr. Şima Uyar.
When I first contacted Jürgen about my plans to write a thesis in Istanbul, he did not hesitate a moment and helped me to get into contact with Şima to arrange everything. Şima, in turn, warmly welcomed me at ITÜ during our first meeting at a summer seminar. And even though it took me more than a year, until I could finally return to Istanbul and start working on my thesis, neither of them lost their patience nor gave up their support.
I have also benefited greatly from my advisors' assistance with regards to our research. Whenever I felt stuck or depressed about my progress, our discussions helped to clear things up, and made things appear in a positive light. Thank you both!

My time in Istanbul was not the first time that I had spend abroad for my studies. I was lucky enough to spend the 2006/07 academic year at the University of Washington, Seattle. Both of these adventures allowed me to find friendship and love all around the world and were unforgettable experiences. I am thankful for those opportunities and for the support I have received during my time abroad, from the German Academic Exchange Service (DAAD), the German National Academic Foundation (Studienstiftung), and from ERASMUS.

Stefan Bach

Zusammenfassung

Evolutionäre Algorithmen (EA) sind Optimierungswerkzeuge, welche auf Darwins Evolutionstheorie und Mendels Genetik basieren. In ihrer über 30-jährigen Geschichte, haben sie sich einen Ruf als gute Löser für schwere Probleme erarbeitet. Diese Diplomarbeit betrachtet die Skalierbarkeit von EAs und ihre Anwendbarkeit auf große Probleme. Literatur zu diesem Thema wird in vier Gruppen vorgestellt:

– Ansätze zur Verbesserung allgemein anwendbarer EAs, welche auf einer Hypothese über die den EAs zu Grunde liegende Theorie basieren (Building Block Hypothese);

– parallele EAs, welche die Ausführungszeit unter Einsatz zusätzlicher Hardware verbessern;

– EAs die problemspezifische Operatoren verwenden; und

– mehrstufige Systeme, welche EA beinhalten.

Die Ansätze der ersten beiden Gruppen haben flexible Algorithmen zum Ziel, welche leicht auf eine Vielzahl von Problemen angewendet werden können. Die beiden letztgenannten Ansätze opfern diese Flexibilität zu Gunsten verbesserter Performanz auf einem spezifischen Problembereich.

Diese Arbeit untersucht experimentell die Skalierbarkeit von evolutionären Clustering-Algorithmen. Clustering Probleme sind allgemein und auch speziell zur Untersuchung von EAs von Interesse. Der allgemeine Reiz von Clustering liegt in der verbreiteten Anwendung in vielen Wissenschaftsbereichen; das Clustering Problem ist nicht nur auf die Informatik beschränkt. Diese Arbeit betrachtet Clustering basierend auf paarweisen Ähnlichkeiten, welches die allgemeinste Modellierung des Problems ist. Für die Analyse von EAs ist Clustering auf Grund der verwendeten Repräsentation von Lösungskandidaten interessant. Die gewählte Kodierung führt zu einer hohen Abhängigkeit zwischen vielen Variablen innerhalb eines Lösungskandidaten; dies erhöht die Schwierigkeit des Problems für EAs.

Zur experimentellen Skalierbarkeitsanalyse sind skalierbare Testdaten notwendig, welche als dünn-besetzte paarweise Ähnlichkeitsmatritzen erstellt werden. Ein einfacher EA, welcher als Referenz eingeführt wird, zeigt schlechte Skalierbarkeitseigenschaften für diese Probleme. Mit wachsender Problemgröße nimmt die Laufzeit schneller als quadratisch

zu. Schon für ein Problem mit 2.000 Objekten beträgt die durchschnittliche Laufzeit bis zum Erreichen zufriedenstellender Lösungen über 20 Minuten. Dadurch ist der Referenzalgorithmus für große Probleme nicht geeignet.

Verschiedene Erweiterungen des Referenzalgorithmus werden vorgeschlagen. Diese integrieren problemspezifisches Wissen in Form von speziellen Rekombinationsoperatoren und durch die Hybridisierung mit Cluster-Heuristiken. Insgesamt ergeben sich durch Kombinationen der vorgeschlagenen Operatoren 126 verschiedene Algorithmenkonfigurationen, welche für Probleme mit bis zu 2.000 Objekten getestet werden. Als Ergebnis der Experimente lässt sich feststellen, dass eine intelligente Initialisierung alleine, ohne Hybridisierung und mit Standard-Rekombinationsoperatoren, keine verbesserte Skalierbarkeit erreichen kann. Es finden sich aber Algorithmen, welche durch Cluster-basierte Rekombination oder durch die Hybridisierung mit einem hill-climbing Algorithmus. So ist es möglich, Probleme mit 2.000 Objekten durchschnittlich in unter drei Sekunden zu lösen. Es werden Laufzeiten erreicht, die fast linear mit der Problemgröße skalieren. Probleme mit bis zu 100.000 Objekten werden mit einer durchschnittlichen Laufzeit von deutlich unter 1.000 Sekunden gelöst.

Die Algorithmuskonfigurationen die mit guter Performanz gemessen wurden werden im nächsten Schritt erweitert. Die Verbesserungen basieren auf bekannten zweistufigen Clustering EAs. Die vorgeschlagenen Verfahren clustern in der ersten Stufe ein größenreduziertes Problem mit einem EA. Anschließend wird die berechnete Population verwendet, um den EA der zweiten Stufe zu initialisieren, welcher dann auf dem original Problem arbeitet. Zur Größenreduktion werden zwei Möglichkeiten vorgeschlagen: die Komprimierung des Suchraums durch das Zusammenfassen von Objekten zu Objektgruppen und das Zerlegen des Problems in mehrere kleinere Probleme, welche unabhängig voneinander in der ersten Stufe bearbeitet werden. Die experimentelle Auswertung zeigt, dass der Ansatz mit Objektgruppen Potential zur weiteren Reduzierung der Laufzeit hat, während das Zerlegen des Problems die Laufzeit nicht weiter verbessert. Der Test des zweistufigen Ansatzes mit guten Algorithmuskonfigurationen zeigt eine verringerte Robustheit, da nun manche zuvor erfolgreiche Konfiguration regelmäßig nur lokale Optima erreicht. Andere Konfigurationen hingegen zeigen beträchtliche Verbesserungen der Laufzeit, z.B. erreicht die beste Konfiguration beständig zufriedenstellende Lösungen mit nur 30% der Laufzeit, die ein einstufiger EA in der selben Konfiguration benötigt. Probleme mit bis zu 100.000 Objekten können so mit einer durchschnittlichen Laufzeit von 200 Sekunden gelöst werden.

Abschließend lässt sich sagen, dass für die Clusteringprobleme evolutionäre Ansätze als Basis für erfolgreiche, gut skalierende Methoden dienen können. Dies setzt jedoch die Integration von problemspezifischem Wissen an passenden Stellen voraus. Obwohl ein Standard EA flexibel genug ist, um ohne großen Aufwand für Clustering Probleme angepasst zu werden, sind Standard Operatoren nicht ausreichend um gute Leistung

oder Skalierbarkeit zu erzielen. Der Standard EA ist nicht geeignet zum Lösen großer Probleme.

Sollen große Probleme mit wenig Aufwand gelöst werden, so kann der Entwurf von problemspezifischen Operatoren zu kostspielig sein. Hier empfiehlt es sich den Algorithmenentwurf auf einer problemspezifischen Heuristik aufzubauen, auch wenn die Heuristik anfällig ist nur lokale Optima zu erreichen. In Kombination mit einem EA hat dieser Ansatz trotzdem gute Resultate gezeigt: der hybride Algorithmus ist zum einen schneller als ein Standard EA im Erfolgsfall und zum anderen erfolgreicher als die nicht-hybride Anwendung der Heuristik.

Ist jedoch bestmögliche Leistung ein Hauptaugenmerk, so kann der hybride Algorithmus weiter optimiert werden, indem problemspezifische Operatoren oder zweistufige Verfahren eingeführt werden. Diese Optimierungen können die Leistung nochmals beträchtlich verbessern. Jedoch sind sie aufwändiger zu entwickeln und erhöhen die Fehleranfälligkeit auf Grund reduzierter Robustheit.

Contents

1 Introduction **11**

2 Background and Previous Work **13**
2.1 Introduction to Evolutionary Algorithms 13
2.1.1 From Historic Roots to a General Definition 13
2.1.2 Evolutionary Algorithm Terminology 14
2.2 Introduction to Clustering 15
2.2.1 A Definition of Clustering 15
2.2.2 Traditional Clustering Methods 16
2.2.3 Modern Clustering Methods 17
2.3 Scalability and Performance of Evolutionary Algorithms 17
2.3.1 Building Blocks and Linkage Discovery 18
2.3.2 Parallelization 20
2.3.3 Using Problem-specific Knowledge 21
2.3.4 Multi-level Approaches 22
2.4 Evolutionary Algorithms for Clustering 24
2.4.1 Representations and Operators 24
2.4.2 Hybridization 28
2.4.3 Objectives 28
2.4.4 Large Problems, Scalability and the focus on EA Improvement . . 30

3 Methodology **35**
3.1 Defining the Goal of this Study 35
3.2 Scalable Test Problems 36
3.3 Design of Experiments 37
3.3.1 Reporting Metrics 37
3.3.2 A Base Algorithm 39
3.3.3 Experimental Setup 40
3.4 Summary 43

4 Improving Operators with Problem-specific Knowledge **45**
4.1 Modifications to Initialization, Crossover and Hybridization 45
4.1.1 Alternative Initialization Schemes 46
4.1.2 Alternative Crossover Operators 47
4.1.3 Hybridization Approaches 50

4.2 Experimental Setup . . . 51
4.2.1 System Setup . . . 51
4.2.2 Evaluated Configurations . . . 51
4.3 Experimental Results . . . 53
4.3.1 The Similarity Heuristic Hybridization in Unsuccessful Runs . . . 54
4.3.2 Modified Crossover and Initialization without Hybridization . . . 57
4.3.3 Configurations Resulting in Near-Perfect Success . . . 63
4.4 The Impact on Scalability . . . 66
4.4.1 Using Intelligent Initialization with Traditional Crossover . . . 66
4.4.2 Using Improved Crossover and Hybridization . . . 68

5 Search Space Reduction by Multi-level Approaches 73
5.1 Using Object-Groups to Compress a Problem . . . 73
5.1.1 The Object-grouping Heuristic . . . 74
5.1.2 Finding a Suitable Algorithm Configuration . . . 74
5.1.3 Performance of the Approach . . . 79
5.2 Cutting a Problem by Snowball Sampling . . . 80
5.2.1 The Snowball Sampling . . . 81
5.2.2 The Splicing Heuristics . . . 83
5.2.3 Finding a Suitable Algorithm Configuration . . . 84
5.2.4 Performance of the Approach . . . 86

6 Summary, Conclusion and Outlook 89

A Experimental Results for Chapter 4 91
A.1 The Initial Evaluation of All Configurations . . . 91
A.2 Results for Successful Configurations and Large Problems . . . 96

B Experimental Results for Chapter 5 99
B.1 The Object-grouping Multi-level EA . . . 99
B.2 The Cutting Multi-level EA . . . 100

Bibliography 103

1 Introduction

Evolutionary algorithms (EA) are optimization techniques that are based on the principles of Darwinian evolution and Mendelian genetics. In their history of more than 30 years, they have gained a good reputation for solving hard problems. Nowadays, EAs are applied to a wide variety of problems and are used both in industry and research. A major source of this success is the flexibility of the evolutionary approach; a standard EA can be adapted to many problems with little effort. The result will most likely not outperform a problem-specific optimizer, but the EA has a good chance to yield acceptable results in a feasible time, at least for smaller problem instances.

In this thesis, we focus on the scalability of EAs and their suitability to tackle large problems. Throughout the history of EA research, the topics of scalability and performance have received a considerable amount of attention. However, the underlying theories of EAs have remained incomplete, so most insight on their scalability and on performance improving measures is gained through experimentation. We will review related work in four groups:

- approaches to improve general EAs, based on a hypothesis about EA theory (building block hypothesis);
- parallelized EAs, which speed up execution through additional hardware;
- EAs that include problem-specific operators; and
- multi-level systems that make use of EAs.

Approaches of the first two groups are aimed towards flexible algorithms, which can still be adapted to many problems; the latter two approaches give up flexibility to improve on a specific problem domain.

Our own research on EA scalability will also be done in an experimental manner. We will evaluate EAs on the problem domain of clustering, which is interesting both in general and for EA research. The general appeal of clustering stems from its wide use in many diverse fields, which gives it importance also beyond the area of computer science. We especially focus on pairwise similarity-based clustering, which is the most general model of the problem. With respect to EA analysis, the attractiveness of clustering lies in the encoding of clustering solutions for EA processing. The employed encoding leads

to a high interdependency between many variables, a factor that poses a hard challenge for EAs.

In this thesis, we will evaluate the scalability of EAs on the clustering problem and investigate their suitability to process large-scale problems. We will test a general EA that uses standard operators and design EAs that contain problem-specific modifications. Our findings are that the runtime of the general EA scales at least quadratically with problem size, which renders the approach computationally infeasible for moderately large problems. Using problem-specific knowledge, various algorithms exhibit a near-linear scalability and are able to solve very large problems in feasible time. We will also test the further extension of good algorithms to two-level EAs; this allows an additional runtime reduction of up to 70%.

In Chapter 2 we give an introduction to the topics of EAs and clustering, and review literature on EA scalability in general and on clustering EAs. Chapter 3 details the goals of this study and presents the derived methodology. Our approaches on scalability improvements are presented in Chapters 4 and 5 according to the two problem-specific classes mentioned above. Chapter 6 summarizes and concludes this thesis.

2 Background and Previous Work

We present required background information for the topics in this thesis and review relevant literature. In this chapter, we introduce evolutionary algorithms, the target of our investigation, as well as clustering, the test problem we use to conduct our analysis. We then review relevant literature on two aspects of EAs; our aim is a broad overview of scalability and performance enhancing techniques for EAs and an in-depth survey of previous work in EAs for clustering.

2.1 Introduction to Evolutionary Algorithms

Evolutionary algorithms, or evolutionary computing, subsumes problem solvers rooted in the principles of Darwinian evolution and Mendelian genetics. This section presents the historic roots of EAs, gives a contemporary definition, and introduces the terminology used in the field.

2.1.1 From Historic Roots to a General Definition

The foundation for current EAs had been laid multiple times by multiple researchers, starting in the 1960s and 70s. *Genetic algorithms* (GA) were introduced by Holland [43], *evolutionary strategies* (ES) by Rechenberg [73], and *evolutionary programming* (EP) by Fogel et al. [31]. *Genetic programming* (GP), the fourth traditional EA dialect, was introduced by Koza [55] in the early 1990s.

Most EAs are population-based problem-solving techniques that largely follow the general scheme of selection, recombination, and mutation in Algorithm 2.1. At the beginning of the field, the traditional EA categories mentioned above had distinctive differences regarding specifics of the general EA flow. Each category had its standard answers to questions, such as whether crossover is to be used or not, whether also parent individuals or only offspring are eligible to progress to the next generation, or whether individuals are represented in binary form, as real numbers, or as trees.

The different substreams grew together around the beginning of the 1990s, and nowadays many variants of EAs are blends of the traditional forms. In 1997, Fogel summarized that EAs "involve the reproduction, random variation, competition, and selection of contending individuals in a population. These form the essential essence of evolution, and

Algorithm 2.1: EvolutionaryAlgorithm

```
1  begin
2      Initialize population;
3      Evaluate population and assign a fitness to each individual;
4      while Termination condition is not fulfilled do
5          Select parents from population;
6          Crossover parents to produce offspring;
7          Mutate offspring;
8          Evaluate offspring and assign a fitness to each one;
9          Select individuals for the next generation;
10 end
```

once these four processes are in place, whether in nature or in a computer, evolution is the inevitable outcome". [30]

2.1.2 Evolutionary Algorithm Terminology

Literature about EAs has spawned a colorful terminology, which mixes terms from traditional optimization techniques with biologically inspired ones. On a high level, an EA is a *population*-based search, which denotes that multiple *individuals* are considered simultaneously.

A single individual represents a *candidate solution*, also called a *phenotype*, for the problem at hand. The low-level view of an individual refers to the bare data stored in the population data structure, the so called *chromosome* or *genotype*. The link between the phenotype and the genotype is established by a *representation*, a mapping that states how candidate solutions are encoded into the machine readable chromosome that is processed by the EA.

The genotype of an individual, the chromosome, is usually a complex data structure, such as an array of integers. The single elements of the chromosome are referred to as *genes*, *variables*, or *loci* (singular: locus). The possible values a gene can take are called *allele*.

The EA scheme in Algorithm 2.1 recreates a population in discrete iterations. This is called a *generational* EA, each iteration is called a *generation*. To create the next generation, *parent individuals* are selected from the population and used for *crossover*, also called *recombination*. This process mixes information from multiple parents to create one or more *offspring* or *children*. The offspring undergo *mutation*, which is an operator that (slightly) modifies their chromosome.

A more elaborate description of EA terminology, along with explanatory examples, can be found in introductory textbooks, e.g., [28] by Eiben and Smith.

2.2 Introduction to Clustering

Clustering is a widely used technique with applications in many diverse fields, such as biology, health care, market research, image processing, or data mining [32], just to name a few. This section gives a definition of clustering and presents traditional clustering methods.

2.2.1 A Definition of Clustering

In a recently published book, Xu and Wunsch state about a definition of clustering:

> "Clustering algorithms partition data objects (patterns, entities, instances, observances, units) into a certain number of clusters (groups, subsets, or categories). However, there is no universally agreed upon and precise definition of the term clusters." [89]

In this thesis, we define clustering as the task of partitioning a set of n objects, $V = \{o_1, \ldots, o_n\}$ into k subsets, so-called clusters, such that the partition $\mathcal{C} = \{C_1, \ldots, C_k\}$, $C_i \subseteq V$ fulfills

- $\bigcup_{C_i \in \mathcal{C}} C_i = V$, the clusters cover all objects;
- $\forall i, j : i \neq j \Rightarrow C_i \cap C_j = \emptyset$, clusters do not overlap; and
- $\forall C_i \in \mathcal{C} : C_i \neq \emptyset$, $\mathcal{C}$ contains no degenerate clusters.

Each partition $\mathcal{C}$ of V that adheres to those conditions is a valid clustering, but one usually aims at finding partitions that have certain *beneficial* properties. An exact definition of "beneficial" cannot be given in general, since the perceived quality of a clustering highly depends on the high-level problem to be solved.

The general agreement between most quality measures for clusterings is that all ask for some form of intra-cluster density and inter-cluster sparsity. In an extensive survey of clustering algorithms, Xu and Wunsch state "Most researchers describe a cluster by considering the internal homogeneity and the external separation, i.e., patterns in the same cluster should be similar to each other, while patterns in different clusters should not" [88].

This definition requires a notion of similarity or dissimilarity between the objects in V. Depending on the problem at hand, this notion can be inherent in the problem, e.g., as an $n \times n$ similarity or distance matrix for the objects in V, or the similarity of objects can be determined based on their features, e.g., through calculating the distance of vectors in a d-dimensional Euclidean space.

2.2.2 Traditional Clustering Methods

Traditional clustering methods are divided into *hierarchical* and *partitional* methods.

Hierarchical Clustering

Hierarchical clustering subsumes methods that create a hierarchy of clusterings $\mathcal{C}_1 \ldots \mathcal{C}_n$, where $\mathcal{C}_1$ refers to the trivial clustering of assigning all objects to a single cluster and $\mathcal{C}_n$ is the opposite trivial clustering of assigning each object to its own cluster. An in-between clustering $\mathcal{C}_i$ can be derived from clustering $\mathcal{C}_{i+1}$ by merging two clusters. Hierarchical methods that start with the singleton clustering $\mathcal{C}_n$ and progress through merge operations are called *agglomerative clustering*. Opposite strategies that start with a single large cluster and repeatedly split clusters are called *divisive clustering*.

In practice agglomerative methods are more popular than divisive ones, since they are computationally less expensive: the number of possible cluster pairs for merging is quadratic in the number of current clusters, the number of choices to split a cluster is exponential in the number of contained objects. Most agglomerative methods merge the clusters which are closest with respect to a certain distance function. Examples are *single linkage clustering*, which calculates the distance of two clusters based on the closest objects in the clusters, and *average linkage clustering*, which averages the distance of all objects in one cluster to all objects in the other cluster.

Hierarchical methods have the potential to provide detailed results, e.g., through the representation of the produced hierarchy as a dendrogram, a special binary tree that depicts which clusters have been merged/split and what inter-cluster distance they had. The disadvantage of hierarchical methods is the inflexibility that an object can never be reallocated after it has been assigned. Their computational costs are usually high, since n iterations of at least linear cost are performed, which results in an overall complexity of at least $\mathcal{O}(n^2)$.

Partitional Clustering

As opposed to finding a hierarchy of clusterings, partitional methods optimize a single clustering. The basis for partitional clustering is a *criterion function* that maps each clustering of the data set to a quality value. A partitional clustering algorithm optimizes this function.

The non-trivial clustering that globally optimizes a given criterion function is usually not known. The number of all possible clusterings is exponential in n, and the problem is $\mathcal{NP}$-hard [8]. Thus, traditional partitional methods perform a heuristic search for a clustering that optimizes the criterion function. Most take the desired number of clusters as a predefined input parameter.

A popular partitional method is *k-means clustering*. It operates on a set of real-valued vectors, $V \subseteq \mathbb{R}^d$. K-means is often used with the summed squared error criterion function, that asks to minimize the summed distance of each object to the centroid (average vector) of its cluster. The algorithm is randomly initialized. It calculates all cluster centroids and reassigns each node to the closest centroid. This is repeated until a stable solution is found.

The complexity of k-means is in $\mathcal{O}(nkd)$, which is faster than hierarchical clustering in most cases. But it suffers from drawbacks, such as convergence to a local optimum and inflexibility in the found cluster shapes. For example, the summed squared error criterion gives preference to hyper-spherical clusters.

2.2.3 Modern Clustering Methods

Due to its applicability in many fields, clustering gets much interest from a large number of researchers. The number of papers on clustering has more than doubled from 2000 to 2008 [89] and a great variety of clustering algorithms has been conceived.

For example, current clustering algorithms

– use *combinatorial search methods* to solve clustering, based on both classical methods, such as simulated annealing, and modern nature-inspired methods;

– tackle clustering with *graph-theoretic approaches*;

– employ *neural networks* for clustering;

– cluster based on estimates of the *probability distributions* that underlie the input data;

– use *kernel-based learning* algorithms.

Many algorithms combine traditional methods with newer approaches, such as hybridizing nature-inspired search techniques with variants of the k-means algorithm.

An extensive review of clustering algorithms can be found in [49] by Jain, the most recent comprehensive review is [88] by Xu and Wunsch, and [52] by Kettenring presents a survey of current applications of clustering throughout all fields.

2.3 Previous Work on the Scalability and Performance of Evolutionary Algorithms

The topics of scalability and performance have a long history in evolutionary algorithm research. In the beginning, EAs had spawned hope for general problem solvers that

exhibit good performance over a wide problem range. During the 1980s "The central theme of research on genetic algorithms has been robustness, the balance between efficiency and efficacy necessary for survival in many different environments" [34]. But in the 1990s, the goal of finding a black-box optimizer superior to random search in general was proven impossible with the "no free lunch" theorem [85].

Nowadays, much research on improving the scalability and performance of EAs is still ongoing; even if a holy grail of optimization cannot be found, performance improvements on a smaller problem range are still being desired. Various approaches have been proposed to improve scalability. We present examples of previous work categorized into four groups: EAs that use linkage discovery techniques, parallelized EAs, EAs that include problem-specific knowledge, and EAs that are used as a component in a multi-level system.

2.3.1 Building Blocks and Linkage Discovery

The goal to produce good EAs that are still applicable to a wide problem range has attracted many researchers that follow the *building block hypothesis* introduced by Goldberg [34]. According to the hypothesis, an EA develops good quality solutions in a hierarchical approach: First, short high-quality subsets of the genome, so-called *building blocks* (BB), are evolved. The building blocks then combine and cover larger genome areas with beneficial configurations until a globally optimal individual is found.

The hypothesis has spawned many EAs aimed at allowing good building blocks to be inherited to offspring with only a low probability of being disrupted. The probability of disruption depends on how tightly the genes of a building block are "linked"; a property that is inherent in the used representation and crossover operators. Gene linkage is best understood by an example: Let us assume a 6 bit problem that consists of two independent 3 bit subfunctions $\mathtt{a}_{0\ldots2}$ and $\mathtt{b}_{0\ldots2}$. Two possible encodings of this problem as a 6 bit string are $\mathtt{E}_1 : \mathtt{a}_0\mathtt{a}_1\mathtt{a}_2\mathtt{b}_0\mathtt{b}_1\mathtt{b}_2$ and $\mathtt{E}_2 : \mathtt{a}_0\mathtt{b}_0\mathtt{a}_1\mathtt{b}_1\mathtt{a}_2\mathtt{b}_2$. Under one-point crossover, encoding $\mathtt{E}_1$ has a tighter pairwise linkage between genes of the individual subfunctions than encoding $\mathtt{E}_2$. For example, the probability to disrupt a building block for problem $\mathtt{a}_{0\ldots2}$ is $\frac{2}{5}$ for $\mathtt{E}_1$ and $\frac{4}{5}$ for $\mathtt{E}_2$.

If sufficient prior knowledge about a problem was available, a practitioner could design a representation that explicitly maps related genes to neighboring positions on the genome. This would give coadapted genes a high probability of being inherited together under one-point or many-point crossover. Such prior knowledge is usually not available. To compensate for this, researchers have investigated EAs that discover linkage groups (related genes) automatically and exploit them during crossover.

Munetomo and Goldberg classify techniques for linkage discovery in three groups [66]: direct detection of bias in probability distributions, direct detection of fitness changes by perturbations, and indirect detection along genetic search of building blocks.

1. Examples of the first group are some *estimation of distribution algorithms* (EDA). EDAs are similar to EAs but instead of crossover they generate and sample a probabilistic model. Some EDAs include techniques for linkage discovery; for example, the *extended compact genetic algorithm* (ECGA) [40] tries to minimize the complexity of the probabilistic model, which leads to the summarization of correlated genes. *Bayesian optimization algorithms* (BOA) [71] model probability distribution as a Bayesian network, a representation that relies on conditional probabilities and thus is able to discover correlated genes.

2. Examples of the second group are enhanced *messy genetic algorithms.* Messy EAs encode individuals through movable (gene, allele) pairs, thus removing the physical gene linkage of fixed position genes [35]. The *gene expression messy genetic algorithm* (GEMGA) [5] uses a two-phase linkage-learning process. It first analyzes all individuals through systematic perturbations. Each gene is perturbed randomly to detect if its current value is optimal in the neighborhood. All neighborhood-optimal genes are assigned to an initial linkage set. Genes with a high probability of co-occurrence in the initial linkage sets then build the final linkage sets. The GEMGA crossover combines detected linkage sets of parent chromosomes.

3. Algorithms of the third group do not detect linkage directly but have been shown to develop tight linkage through their representation and operators. The *linkage learning genetic algorithm* (LLGA) [14] is an example of this group. LLGA represents a genome as (gene, allele) combinations that are positioned on a ring structure and interspaced by empty, "non-coding", segments on the ring. Each genome contains all possible (gene, allele) combinations; phenotypes are derived by circling the ring from a random starting point and selecting the first allele value for each gene. Crossover selects a random segment in one parent and grafts it into the other. Studies of LLGA show that the linkage of a building block during crossover increases if the BB is either completely transferred from one parent to the other or not disrupted by crossover in the receiving parent.

An extensive survey of linkage learning techniques can be found in [15]; [76] surveys algorithms of the second and third category, and [51] gives an elaborate non-technical introduction to linkage-learning. Many published studies report favorable results for linkage-learning EAs, such as better performance than simple EAs [40] or linear convergence times for some problems [5][14]. But most test problems that have been used

in those studies are concatenations or hierarchies of independent, small (up to 5 bit) problems. Large problems constructed in this way are especially well-suited to building block approaches, since they are fully decomposable into the single building blocks.

Today, the validity of the building block hypothesis is a controversial topic among researchers. Some defend it with pragmatic reasons, such as "all things are made out of building blocks, whether they be tables, giraffes or computer programs" [80]. Others find "The widespread belief that genetic algorithms are robust by virtue of their schema processing is [. . .] more the result of salesmanship than logical analysis" [84] and note "The various claims about GAs that are traditionally made under the name of the building block hypothesis have, to date, no basis in theory, and, in some cases, are simply incoherent" [86].

2.3.2 Parallelization

For most EAs, parallelization is a straight-forward way to speed up execution, due to an inherently parallel program flow. Most operations require only one or two individuals as input: recombination, mutation, and evaluation of independent individuals can all be performed in parallel. In a generational EA, synchronization is only needed at the beginning of a new generation to select parents and at the end to construct the next generation. Simple parallelization approaches utilize this: A central process manages a single population and distributes parallelizable workload to slave processes. This model is known as the *master/slave model* or *farming model*. It can easily be applied to many (generational) EAs. The possible speedup is limited, according to Amdahl's law [4], by the percentage of serial work, which the central process performs.

Achieving further speedup requires to either parallelize the intrinsic serial tasks [7] or to depart from the general EA scheme towards algorithms that are designed for parallel execution. Algorithms that follow the latter proposal are grouped into coarse-grained and fine-grained models [1][11][74]:

1. Parallel EAs of the *coarse-grained model*, also called *island model* or *deme model*, consist of multiple, largely independent subpopulations [61]. Each computing node manages a single population and executes a regular EA. According to a defined migration strategy, such as "every m generations", the different subpopulations exchange a number of individuals with neighboring subpopulations. This EA variant automatically introduces a niching effect, which prevents a single individual from quickly taking over the whole population, which has been reported as beneficial for population diversity.

2. The *fine-grained model*, also called *diffusion model*, *neighborhood model*, or *cellular model*, is an extreme case of the island model [2]. Each individual is assigned to

its unique processing node. All nodes are connected in a certain pattern, such as a grid, to form a structured population. Each node is self-scheduled in selecting parent individuals from its neighboring nodes, generating offspring, and deciding on the possible replacement of its individual with newly generated offspring.

Unlike the parallelization of traditional EAs, which results in a very predictable speedup, the specialized parallel EAs constitute new evolutionary approaches. Researchers have reported some positive results, such as superlinear speedup, but as is the case with non-parallel EAs, the theory behind those algorithms is far from complete. Implementations of the parallel EA approaches presented above are surveyed in [1], [11], and [74].

2.3.3 Using Problem-specific Knowledge

The "no free lunch" theorem [85] states that all black-box search algorithms have an equal average performance over the space of all possible problems. This means that an EA pays for good performance on a certain class of problems with worse performance on other types of problems. This is acceptable if one looks for an EA with good performance and scalability for a limited problem range. In this case, performance can be actively tuned towards a certain problem-type with the inclusion of problem-specific knowledge in the EA.

The EA flowchart (Algorithm 2.1) allows many possibilities to include problem-specific knowledge. We present four categories by example, depending on the use of informed initialization, informed operators, a problem-specific representation, and hybridization with traditional techniques.

1. The impact of a *problem-specific initialization* is analyzed in [81] for four real-world problems. The authors derive initial populations from good solutions that have been found by heuristics. A good solution is mass-mutated with varying mutation rates (from 0% to 100%) and used to seed the initial population. A 0% mutation rate corresponds to a population that is seeded with identical copies of the good solution, a 100% mutation rate corresponds to a completely random initialization. The authors find that low mutation rates for the mass-mutation result in a higher average fitness of the best individual. But with lower mass-mutation rates, variance in the fitness of the best individual is also reduced. For some problems, the overall best result was only obtained with a purely random initial population.

2. An example for EAs that use *problem-specific operators* is the algorithm for a multi-day scheduling problem given in [23]. In the paper, the problem is formulated as an *integer linear program* (ILP). The authors first compare a traditional ILP

solver with a standard EA that uses a random initial population, binary-coded individuals, one-point crossover, and bitwise mutation. The traditional ILP solver fails to find feasible solutions for problems of 200 variables. The standard EA is able to find feasible solutions for up to 300 variables but shows an exponential growth in the number of required function evaluations. The EA is then enhanced with problem-specific operators: e.g., for each day, crossover chooses the schedule of the parent with higher resource utilization; mutation modifies infeasible solutions to become feasible. The problem-specific EA is successfully applied to problems of up to 1 million variables and exhibits sub-quadratic growth of computational time.

3. A step further than just problem-specific operators goes the introduction of a *problem-specific representation*. An example is the EA for a spanning-tree problem in electrical distribution planning presented in [12]. The authors find drawbacks in two early EA approaches, which both represent solutions as straightforward binary strings over the possible tree edges; included edges are marked with 1s. Traditional operators create many infeasible solutions in this case, since only a small subset of the binary strings represents valid tree structures. This is countered by a crossover that builds offspring as spanning trees in the union of parental tree edges. The authors claim that this improvement does not support timely convergence, since large structures in the parent trees are unlikely to survive recombination. They suggest to represent spanning trees by storing direct precedence relationships between nodes as seen from a fixed root. A novel crossover selects random paths in one parent and injects them to the other in a feasibility preserving manner. Experimental evaluation for a fixed number of generations shows that this novel approach has, on average, identified twice as many correct edges than the traditional EA. The result of the improved EA with a standard representation lies in between.

4. The furthest-reaching extension to traditional EAs is a *hybridization with problem-specific techniques*, which yields so-called *memetic algorithms* [69]. Newly created offspring can be subjected to a variation of classical optimization approaches, ranging from hill-climbing improvement to highly specialized problem-dependent heuristics. Memetic algorithms for clustering problems will be discussed in Section 2.4.2, further examples for a wide range of real-world applications can be found in [41].

2.3.4 Multi-level Approaches

The approaches mentioned so far, which are used to improve scalability and performance, amend the inner workings of an EA. A different direction is taken by researchers that

modify the outside environment around EAs and use them only as part of a multi-level problem solving workflow. We group the presented examples in two-level EAs, EAs as part of a sequential workflow, and EAs as high-level controllers in hyper-heuristics.

1. *Two-level EAs* have been introduced in situations where a given EA was computationally infeasible for large problem sizes. Section 2.4.4 will describe example two-level EAs for the clustering problem in more detail: An algorithm by Gündüz-Öğüdücü and Uyar performs a first-level EA run on a compacted version of the given problem and then unpacks the results to seed a second-level run; a clustering algorithm by Korkmaz uses an EA first on partial versions of a problem and then builds a complete solution from found subsolutions in a second EA run.

2. An example that uses an EA only for *one level in a sequential workflow* is given in [44] for a vehicle routing problem with time windows; an optimization problem aimed at both minimizing the number of vehicles and the total traveled distance. The problem is usually approached by traditional search techniques, but the authors find that for large problems, a neighborhood search is unlikely to reduce the number of vehicles, since this requires extensive changes to a given solution. They propose a two-level approach that first optimizes the number of vehicles using an EA and then optimizes the traveled distance of the best solution with a tabu search. Experiments place the proposed algorithm "among the best methods" for smaller problems and show that it is "very competitive for very large problem instances".

   A reversed approach is introduced in [17] where the authors first employ a heuristic and use an EA in the second level. The paper deals with a problem in airline crew scheduling: the planned trips of one month need to be assigned to the employees schedules, so that all trips are covered and constraints on the schedules are met. Each schedule should have a high workload, as to cover the trips with the minimum number of employees. The authors first build as many complete, high quality schedules as possible with a heuristic. This procedure leaves a number of open trips, which the heuristic failed to assign. In the second level, an EA is used to combine the leftover trips into schedules. This approach reduces the time needed in the EA-phase, since the EA only deals with a limited subset of the initial problem.

3. Recent approaches combine EAs and traditional heuristics in a hierarchical manner, to obtain so called *hyper-heuristics* [9]. Unlike common meta-heuristics, which operate on the space of solutions, a hyper-heuristic works on the space of low-level heuristics developed for the problem. One possible variant employs EAs as the high-level search. An example for this approach is shown in [21] for a trainer scheduling problem. The system uses 12 low-level heuristics, and the EA evolves a sequence for their application in order to construct a solution. The authors show

in experiments that the hyper-heuristic system outperforms both a traditional EA and the sole application of the low-level heuristics.

Other authors use genetic programming to design heuristics from scratch. Both [10] and [57] find the evolved heuristics for bin-packing and a version of the knapsack problem to be on par with human-designed heuristics.

2.4 Previous Work on Evolutionary Algorithms for Clustering

Clustering problems, especially those that have traditionally been solved with partitional clustering methods, can be seen as combinatorial optimization problems with the goal to find an optimal solution to the criterion function. Clustering optimization problems are known to be $\mathcal{NP}$-hard [8]. According to current knowledge, this renders it impossible to reliably find an optimal solution in an acceptable amount of time.

Evolutionary algorithms have earned themselves a reputation as a good heuristic for solving hard optimization problems. Thus it is little surprise that they have been applied to clustering problems. First clustering EAs appeared at the beginning of the 1990s as adaptions of the simple genetic algorithm to the clustering problem [56] [6]. Since then, a variety of implementations has been developed, and more sophisticated representations and operators, as well as hybridizations with other search techniques have been introduced.

Cole gives one of the first reviews of EAs for clustering in her thesis [18], a survey on metaheuristics for clustering by Rayward-Smith [72] includes several EA approaches, and Hruschka et al. present a recent survey of EAs for feature-based clustering [47], which is an extension of their previous work [68]. The majority of previous work deals with feature-based clustering, especially for clustering vectors in $\mathbb{R}^d$. Thus, in this review, we specifically note algorithms that can handle similarity or distance information directly.

This section gives a short overview of EAs for clustering. We survey various representations and their genetic operators, describe hybridization approaches that combine EAs with other techniques, discuss possible objective functions, and give special consideration to approaches that either focused on large problems or scalability improvements.

2.4.1 Representations and Operators

The basis for designing an EA is the representation that maps the phenotypical problem to a genotypical data structure. The chosen representation then constraints the range of standard genetic operators for crossover and mutation, and is the basis for custom

problem-based operators. The majority of clustering EAs uses direct or centroid-based representation, but medoid-based representations and others have been used as well.

Direct Representations

A direct representation, sometimes called label-based representation or group number encoding, explicitly stores the assignment of each object to a cluster. Due to its simplicity the direct representation is used by most early work, e.g., in the form of bit strings to group data into two clusters [56] or as $n \times k$ binary matrices that have a single 1 in each column [6]. In later work, integer strings of length n are more common; they store a cluster number $1 \dots k$ for each object [75] [26].

The papers referenced above employ standard genetic operators; they recombine two parents through one-point, many-point, or uniform crossover and perform mutation through bit-flips or random reassignment of group numbers. A drawback of this combination is the resulting context-insensitivity: two individuals that represent equal solution candidates, such as `11122|2233` and `22233|3311` for a problem with 9 objects, can produce a different offspring, e.g., `111223311` for one-point crossover at the marked position. To counter this, some authors use normalization in their algorithms, e.g., by permuting the group numbers in one parent to match those of the second during crossover [58] [64]. A positive effect of this strategy on solution quality is found by Choi and Moon [16]. Kim et al. [53] formalize the normalization process through the introduction of a labeling-independent distance between two individuals and a geometric crossover based on this distance.

Hruschka et al. use special crossover and mutation operators with a direct representation in their *clustering genetic algorithm* (CGA) and *evolutionary algorithm for clustering* (EAC) [46]. CGA employs a crossover similar to the one presented by Falkenauer in [29]: a subset of clusters in one parent is chosen and copied into the second parent. In the second parent "the unchanged clusters [...] are maintained and the changed ones have their unaffected objects allocated to the corresponding nearest clusters (according to their centroids)". EAC knows two problem-specific mutations. The first one splits a single cluster into two, assigning membership based on the distance to initially selected seeds. The second eliminates a cluster, assigning its objects to the remaining clusters based on their distance to cluster centroids.

Uyar and Gündüz-Öğüdücü also use a crossover that is independent of the labels stored in a direct representation [83]. The object-focused operator randomly selects an uncovered object and a parent. The selected object and all other uncovered objects in its cluster then form a new cluster in the offspring. This process is repeated until all objects are covered.

Most algorithms that use a direct representation are able to handle similarity-based clustering, even if they have not been initially designed to do so. Exceptions of this are algorithms that use cluster centroids in their criterion function or that employ special centroid-based operators, such as CGA and EAC.

Centroid-based Representations

A centroid-based representation stores k feature vectors for a k-clustering problem, called centroids. The assignment of objects to clusters is then indirectly determined by assigning each object to the closest feature vector. Common to most approaches is to store the centroid vectors coded as real numbers, e.g., [13], [59], and [63], but a variety of genetic operators is used.

When treating each feature vector as a single locus, traditional two-point [13] and uniform crossover [63] can be applied. In order to increase the likelihood of well-distributed centroids in the offspring, [13] sorts the genotype according to the coordinates of the first dimension. Linear crossover in [59] combines two parent centroids linearly to produce an offspring centroid. Merz and Zell suggest a special distance-based crossover, termed replacement recombination operator in [63]: For two parents a and b, they match the centroids in parent b to those in parent a according to their distances and replace unmatched, "far-away" centroids in parent a randomly with centroids of parent b that are not exclusively assigned.

A commonly used mutation operator for real coded centroids is Gaussian perturbation [13] [59], which randomly moves the centroid in the search space. Merz and Zell suggest a distance-based mutation [63], which creates a new centroid by searching a random cluster for the object most distant to the center; this object replaces a randomly selected centroid.

A more exotic centroid-based approach is proposed by Xiao et al. in [87]. They encode centroid coordinates in quantum-inspired Q-bits, which hold probabilities for each bit to be zero or one. Their algorithm repeatedly collapses the Q-bits into a binary state, runs several generations of a standard centroid-based algorithm, and "rotates" the Q-bits so that "the use of quantum-gate rotation is to emphasize the searching direction toward" the best individual.

Medoid-based Representations

Medoid-based representations are similar to centroid-based ones, but have the additional constraint that cluster centers have to coincide with an object $o \in V$. This allows a simple representation by listing the index of each medoid object. Such a representation is used in [60] and [77]. The crossover operator pools the parent medoids and divides them again randomly to create two offspring. Mutation is performed through random

replacement of medoids. The same representation is used in [65] but with traditional one-point crossover.

Medoid-based methods are suitable for similarity-based clustering. Unlike centroid-based methods that define cluster centers as unconstrained coordinates, medoid-based methods fix the centers to representative objects. But unlike direct encoding, which explicitly assigns each object to a cluster, medoid-based methods still require a decoding step to produce the final solution. This decoding requires a full similarity matrix in order to compare each object to every medoid. If only sparse similarity data is available, medoid-based algorithms are not suitable.

Others

Other than the above mentioned representations, some less common methods are used for EA clustering as well. Speer et al. base their algorithm on a *minimum spanning tree* (MST) of the data set [79]. Solutions are represented by "deleted edges" in the MST, which then induce a clustering through the remaining connected components. Crossover is done by pairing up the deleted edges of two parents randomly and passing on the deleted edge of either parent with equal chance. Mutation randomly exchanges a deleted and a non-deleted edge. In their original work, the EA is used for feature-based clustering, but this approach is also suitable for similarity-based clustering, if the data set allows to create an MST.

Another less common representation is based on linkage encoding, which is used in [27], [39], and [54]. In this encoding, individuals are strings of n object indexes, with the interpretation that the entry j on the i-th position corresponds to a link of o_i to o_j. A clustering solution is calculated based on the connected components in the induced graph. Similar to a direct encoding, this representation maps many genotypes to the same phenotype. Both [27] and [54] allow forward links only and limit the number of incoming edges to 1 for each node that does not point to itself. This results in a one-to-one mapping of genotypes and phenotypes.

Linkage encoding allows the use of traditional one-point and uniform crossover. Traditional operators are reported to work well, since they pass on much structure of the parent clusters to the offspring while merging and splitting clusters automatically. This has been used for algorithms that perform a search for the correct number of clusters, since solutions with variable k can be easily represented with a fixed-length genome. The assignment of each node to a cluster can be derived from the linkage encoding without involving distances or similarities. Thus the resulting algorithms are generally able to deal with sparse similarity-based clustering, as long as the criterion function is compatible as well.

2.4.2 Hybridization

Several previous approaches introduce further methods to improve the algorithms described above. A common strategy to achieve better results and/or faster convergence is the hybridization of an EA with local search methods or traditional heuristics.

Out of the above algorithms, [26], [75], and [79] introduce neighborhood hill-climbing to optimize the produced offspring; small perturbations, such as swapping deleted and non-deleted MST edges or reassigning single objects, are tested for their effect on the overall fitness. If they have a positive effect the update is accepted, otherwise it is reversed; this procedure can be repeated several times. The EA in [75] is also amended with a tabu list that prevents the algorithm from considering individuals again that have already been evaluated before.

Other EAs for clustering have been hybridized with traditional partitional clustering techniques. For example, [46] and [63] use k-means iterations to optimize centroids or to produce an updated labeling for a direct encoding.

2.4.3 Objectives

The suitability of an objective or criterion function in a clustering EA is highly dependent on the problem at hand. No general function exists that is able to differentiate a “good” clustering from a “bad” one, since even the interpretations of good and bad are usually problem dependent. Each criterion function usually gives a certain preference to some special form of solution. For example, single-linkage clustering often results in elongated clusters, k-means clustering and other algorithms that are based on the summed squared error criterion lead to spherical clusters.

Important for choosing a proper objective function is also the question whether the clustering algorithm takes a fixed number of clusters, k, as an input parameter or needs to determine the best value automatically. The former usually poses no problems, but the latter approach needs to consider certain aspects of the criterion function. Most functions have their global optimum in one of the trivial clusterings, e.g., the summed squared error equals zero for $k = n$.

To find a good k, independent of the preferences of a single criterion, some previous work uses *multi objective evolutionary algorithms* (MOEA); algorithms that discover a front of Pareto optimal solutions. Most MOEAs for dynamic-k clustering in the literature employ two contradicting objectives so that one is optimal for singletons and the other one for a single cluster. Multi objective clustering EAs are based on traditional MOEAs, such as the clustering EAs by Du et al. [27] based on NPGA [45], Chen and Wang [13] and Mukhopadhyay and Maulik [65] based on NSGA-II [22], and Handl and Knowles [39] based on PESA-II [20].

Demir et al. compare multiple variants of MOEAs for clustering in [24], including the algorithm of Handl and Knowles and a version of [83] that is adapted to work with the SPEA2 [90] multi objective algorithm. Their study presents various objective functions, which we summarize as a good example of the existing variety.

Min-max-cut (MMC) sums the ratio of inter-cluster similarity over intra-cluster similarity for each cluster, based on a similarity function s. It is to be minimized and reaches its global optimum for $k = 1$ cluster:

$$\mathrm{MMC}(\mathcal{C}) = \sum_{C_i \in \mathcal{C}} \frac{\mathrm{cut}\,(V \setminus C_i, C_i)}{\sum_{o_x, o_y \in C_i} s\,(o_x, o_y)} \tag{2.1}$$

$$\mathrm{cut}\,(V \setminus C_i, C_i) = \sum_{o_x \in C_i} \left(\sum_{o_y \in V \setminus C_i} s(o_x, o_y) \right)$$

Connectivity (CO) is also optimal for $k = 1$ cluster and is to be minimized. It sums up penalty values by considering the L nearest neighbors of each object, with $\mathrm{NN}_{o_x}(j)$ denoting the j-th nearest neighbor of o_x, and assigns a penalty if they lie in separate clusters:

$$\mathrm{CO}(\mathcal{C}) = \sum_{o_x \in V} \sum_{j=1}^{L} \begin{cases} \frac{1}{j} & , \nexists C_i \in \mathcal{C} : o_x \in C_i \wedge \mathrm{NN}_{o_x}(j) \in C_i \\ 0 & , \text{otherwise} \end{cases} \tag{2.2}$$

Overall deviation (OD) considers the distance of objects to their cluster medoids μ, based on a dissimilarity function d. It is to be minimized and reaches its global optimum for $k = n$ clusters:

$$\mathrm{OD}(\mathcal{C}) = \sum_{C_i \in \mathcal{C}} \sum_{o_x \in C_i} d\,(o_x, \mu_i) \tag{2.3}$$

Global silhouette (GS) is also based on dissimilarities. The index calculates a silhouette value for each object o_x, based on comparing the average dissimilarity of o_x to its own cluster, $\mathrm{a}\,(o_x)$, with the average dissimilarity to the next similar cluster, $\mathrm{b}\,(o_x)$. The value is to be maximized and optimal for $k = n$ clusters:

$$\mathrm{GS}(\mathcal{C}) = \frac{1}{k} \sum_{C_i \in \mathcal{C}} \left(\frac{1}{|C_i|} \sum_{o_x \in C_i} \mathrm{sil}\,(o_x) \right) \tag{2.4}$$

$$\mathrm{sil}\,(o_x) = \frac{\mathrm{b}\,(o_x) - \mathrm{a}\,(o_x)}{\max\left\{\mathrm{a}\,(o_x), \mathrm{b}\,(o_x)\right\}}$$

$$\mathrm{a}\,(o_x) = \begin{cases} \frac{1}{|C_i|-1} \sum_{o_y \in C_i \setminus o_x} d(o_x, o_y) & , |C_i| > 1 \\ 0 & , \text{otherwise} \end{cases} \qquad (o_x \in C_i)$$

$$\mathrm{b}\,(o_x) = \min_{C_j \in \mathcal{C} \setminus C_i} \left\{ \frac{1}{|C_j|} \sum_{o_y \in C_j} d(o_x, o_y) \right\} \qquad (o_x \in C_i)$$

Further objective functions have been used in other studies, but not all of them are compatible with similarity-based clustering. Minimizing the *number of clusters* is the most simple objective to counteract a criterion function that reaches its global optimum for $k = n$ clusters. Within-cluster variation functions are optimal for $k = n$ clusters, e.g. the *total within-cluster variation* (TWCV). TWCV sums the squared distances of all objects to their cluster centroids. Other within-cluster variation functions use, e.g., distances instead of squared distances or medoids instead of centroids.

2.4.4 Large Problems, Scalability and the focus on EA Improvement

So far we have presented a general survey of clustering EAs. We now focus on previous work relevant for researching and improving the scalability of clustering EAs. We group this work into publications that focus on large problem sizes or introduce special measures to handle large problems on the one hand and publications that focus on the systematic comparison of various clustering EAs on the other hand.

Clustering Large Problems with EAs

Both Jie et al. [50] and Gasvoda and Ding [33] use clustering EAs with very large data sets. Both studies use a prototype (centroid) based representation, which allows for a chromosome length that only depend on the number of clusters k and the number of features per item, but not on the number of objects n.

Jie et al. use an EA that clusters objects with mixed numerical and categorical feature data, which they test with randomly generated objects. The largest data set consists of 80,000 objects with 9 numerical and 11 categorical features. Independent experiments for varying both the number of clusters in the test data and the number of objects are performed and a linearly dependent CPU time is found for both cases. The paper lacks information regarding the termination condition that determines at which point in the EA run the presented CPU times are measured. Experiments have been repeated only 5 times, which is a rather small number for the evaluation of a probabilistic algorithm.

Gasvoda and Ding use an EA to cluster large numerical data, which they test for up to 250,000 objects. Based on an EA that uses a standard centroid-based representation and traditional genetic operators, they introduce preprocessing of the data set to reduce the time required for calculating a k-means like fitness function. Their work tests two preprocessing strategies: random sampling and summarization. In the first case the fitness calculation is based on a random subset of the input data, in the second case a grid is laid on the feature space and a weighted summary-vector is calculated for each grid cell, based on the feature vectors contained in the cell. The resulting algorithms are evaluated for a fixed number of generations and compared to each other as well as

to a traditional k-means algorithm. They find that the new EAs produce better quality results than pure k-means and that random sampling beats the summary scheme.

Tasoulis and Vrahatis [82] introduce a special objective function with the intention to speed up EA clustering. In standard EAs most computational effort is spent on evaluating the objective function for each individual in each generation. To reduce this effort their work presents the *window density function* (WDF), an objective suitable for feature-based clustering. An individual is represented through k window centers around which a hyper-cube of fixed size is imagined. The fitness of an individual is then derived from the summed density of objects inside all windows, a calculation that can be performed in sublinear time with orthogonal range search techniques. Their work presents several differential evolution algorithms that employ WDF and compares the achieved results to a traditional clustering technique. The experimental evaluation of the performance improvement or a comparison to other EA-based clustering techniques is not part of the publication.

Clustering EAs by Gündüz-Öğüdücü and Uyar [37] as well as by Korkmaz [54] introduce reduction techniques to cope with large problems. Both works are based on representations that require individuals of length n (direct and linkage-based) and see the necessity to reduce problem sizes to allow efficient processing. A comparison of the resulting two-level EAs to unaltered algorithms is not discussed in either case.

Gündüz-Öğüdücü and Uyar reduce the search space by combining multiple objects heuristically; all objects are merged with their nearest neighbors. The compressed problem is then clustered by an EA as stage one of the overall process. Based on the resulting clustering for the compressed problem, an initial population for the original problem is derived by unpacking the combined nodes. A normal EA run is conducted as the second stage to further refine the node assignment.

Korkmaz uses a similar approach that consists of a two-level clustering EA. First the data set is randomly divided into s non-overlapping subsets. A multi-objective EA is used to cluster each subset independently, which generates s Pareto fronts as the result of stage one. In the second stage the optimal number of clusters k in the data set is estimated using the elbow criterion on the Pareto fronts. A k cluster solution is selected for each subset and the $s \cdot k$ resulting clusters are then processed in a second EA run to decide which clusters should be combined.

Comparing Improvements to Clustering EAs

Several works focus on comparing either EA methods to different clustering techniques or on comparing different improvement ideas implemented in a basic EA for clustering. We are interested to see how experiments in those cases are designed and what results are reported.

Paterlini and Krink [70] experimentally evaluate various methods for medoid-based clustering. They test a GA-variant, particle swarm optimization, and differential evolution, as well as k-means and random search. The test problems for their study are taken from the popular UCI machine learning repository with problems up to 870 objects. To compare the various approaches, the number of fitness evaluations for each algorithm is set to a fixed limit of 100,000 and the resulting solution qualities are compared. The study shows an advantage for the differential evolution algorithm over the others, which reached the best result for all test data.

Sheng and Liu [77] introduce a *hybrid k-medoid algorithm* (HKA) which contains a k-means like search heuristic: after applying the regular genetic operators, each individual is optimized by exploring the nearest neighbors of the current medoids as alternative cluster centers. The optimization is performed in a hill-climbing sort of fashion and accepts only improvement steps. The work evaluates HKA on biological gene expression data sets of up to 3,000 objects and compares it to older non-hybridized EA approaches. Experiments record the run-time until convergence and the solution quality for a fixed number of fitness evaluations. Results show that HKA converges much faster and also reaches a better final fitness than its competitors. But the latter finding needs to be viewed with caution, since the comparison algorithms are judged by the fitness measure of HKA even though they are guided by their original objective functions during the search.

Dias and Ochi [26] amend a basic direct encoding EA with various improvement ideas. They add a hill-climbing hybridization to the best individual, which they execute once or for repeated times; introduce the possibility to increase the number of clusters in order to find better quality solutions; and test resetting the population, based on mass-mutations of the best individual, every time a new best individual is found. In total, seven algorithms are created in addition to the basic algorithm, each using various combinations of the introduced improvements. All algorithms are evaluated by running for a fixed number of iterations on artificial problems of up to 500 objects. A clear winner cannot be found, since no algorithm achieves the best performance for all tested problem sizes. But the authors show that each improved version (all of them using some form of hill-climbing) outperforms the basic algorithm.

Alves et al. [3] work on improving the performance of the EAC algorithm presented earlier. They propose various improvement ideas for the baseline algorithm, but initially do not combine them. Improvements to mutation adjust the probabilities of the split and eliminate operators based on either cluster quality or on past operator performance. When eliminating a cluster, the baseline algorithm assigns each element to the closest centroid, this scheme is simplified by adding all nodes to the same target, which saves computational time. The modified algorithms are evaluated on artificial and biological problems of up to 900 objects. The termination condition is either the achievement of a

reference fitness or a fixed generation limit, whichever happens first. The authors report the CPU time used by each algorithm until termination but give no details on which cause triggered the termination. An ANOVA analysis is performed to compare the results of different data sets and fitness functions but no specifics on the analysis are reported. The changes that showed statistically significant improvements are combined in a new *fast EAC* (F-EAC) algorithm. The evaluation of F-EAC shows that it outperforms all other options.

3 Methodology

We define the goal of this study and decide on the experimental methodology. In this chapter we first explain the goal of this study based on our understanding of scalability, we then introduce our design for scalable test problems, and finally we conceive the detailed experiment design. The last section summarizes the resulting methodology.

3.1 Defining the Goal of this Study

Evolutionary algorithms have become known as good optimizers for $\mathcal{NP}$-hard problems, but their underlying theories are still incomplete. Thus, most insight on EAs is gained through experimentation. But many researchers restrict their experiments to relatively small genome sizes of a couple of hundred variables. Quite often, experiments are only performed for a single problem size. Such setups allow no insight on the scalability of newly proposed methods. This raises our interest in the suitability of EAs to efficiently search on very large genomes.

For the purpose of this study, we define scalability as the *behavior of performance over changing problem sizes*. Due to the probabilistic nature of EAs, "performance" allows for a variety of interpretations; such as the achieved solution quality, required run-time, resource usage, and the probability to find a satisfying solution. We focus mainly on run-time, the traditional performance metric used to evaluate deterministic algorithms. Our detailed reporting metrics are presented in Section 3.3.1.

Achieving a general insight on EA scalability is a desirable goal. But without a sound theoretical background, scalability can only be evaluated experimentally, which limits us to a small problem range. Previous research has used test problems that have been especially designed to fit the analyzed algorithms, such as problems constructed from many small building blocks, which we described in Section 2.3.1. We decide to investigate clustering problems, since they allow for large-sized problems that have a high interdependency between many variables. For example, under a direct encoding, the move of a single object to a different cluster changes its relation to all other objects in both the old and the new cluster. Also, clustering is important beyond the area of computer science because of its wide use in many diverse fields. While many clustering algorithms only deal with feature-based data, we decide to use problems that are given

by pairwise similarities among objects in a data set. This is a more general model of clustering and allows to process feature-based and similarity-based data alike.

In short, the goal of this study is an analysis of EAs for the clustering problem that focuses on the behavior of run-times over changing problem sizes.

3.2 Scalable Test Problems

To conduct experiments over changing problem sizes, we need test problems of variable sizes. We see two requirements for suitable test problems: In order to meaningfully compare algorithm runs on different problem sizes, all test problems should exhibit similar characteristics that scale well with size. In order to rate the quality of an achieved solution, an optimal or nearly optimal solution should be known. We know of no real-world data set for clustering that fits the required conditions and thus we decided to create artificial test problems.

Our problem generator probabilistically builds sparse similarity matrices. The employed algorithm is similar to a graph generator by Gündüz-Öğüdücü and Uyar [37]:

Algorithm 3.1: ProblemGenerator

```
1  begin
       Input: int n, int k
2      Assign each object to a random cluster 0...k − 1;
3      for 5n times do                                        /* Add sim. entries: */
4          obj1 ← Choose a cluster and one of its objects randomly;
5          if (Random number in [0, 1]) ≤ 0.7 then           /* intra-cluster */
6              obj2 ← Choose a different object in the cluster of obj1 randomly;
7              Create an intra-cluster entry for obj1, obj2;
8          else                                               /* inter-cluster */
9              obj2 ← Choose a different cluster and one of its objects randomly;
10             Create an inter-cluster entry for obj1, obj2;
11     foreach object 0...n − 1 do                            /* Check intra-cluster sim. */
12         while intra-cluster similarity sum < inter-cluster similarity sum do
13             Create an intra-cluster entry for the object and a random partner;
14 end
```

The first step randomly assigns n objects to k clusters. We then add similarity entries for randomly chosen object pairs to the similarity matrix. Finally, we run a check procedure that guarantees a minimum intra-cluster similarity per object. When adding an entry to the sparse similarity matrix, we randomly decide whether an intra-cluster (70%) or inter-cluster (30%) connection should be created and draw a matching object

pair randomly. We chose the same ratio of intra-cluster to inter-cluster entries as in [37]. The similarity values are drawn from a normal distribution with $\mu_{\text{intra}-\text{cluster}} = 0.75$, $\mu_{\text{inter}-\text{cluster}} = 0.25$. We chose $\sigma = 0.25$ to generate some overlap between the sampled intra-cluster and inter-cluster values. A too small standard deviation would result in clusters that could be easily separated. If values outside the range $[0, 1]$ are sampled, they are discarded and redrawn.

The problem generator adds $5n$ similarity entries to each cluster problem, which corresponds to an expected node-degree of 10 in the induced similarity graph. This scales the number of non-zero entries in the similarity matrix linearly in the number of objects. Thus, problems of different sizes exhibit equal properties regarding the expected number of intra-cluster and inter-cluster similarity entries as well as their similarity values.

The generation process does not assure that the initial clustering chosen in the first step is the best clustering for the resulting problem. But the initial assignment drives the problem generation: intra-cluster entries are created with a higher probability and higher expected similarity than inter-cluster entries. Thus, the initial clustering is likely to achieve intra-cluster density and inter-cluster sparsity. To further increase this likelihood, the final step in Algorithm 3.1 asserts that for each object, the summed intra-cluster similarity is at least as high as the inter-cluster sum. Otherwise, additional intra-cluster entries are added.

To guide a search on the generated problems, we need to decide on a criterion function. From the list of functions presented in Section 2.4.3, we select the min-max cut (Equation 2.1), which considers both intra-cluster and inter-cluster similarities. For easier visualization, we set our objective function to $\frac{1}{1+\text{MMC}}$ to obtain a maximization problem. For easier comparability, we report values of the objective function relative to a reference value; the reference fitness is the objective value achieved by the initial clustering during problem generation.

3.3 Design of Experiments

We have set the goal of this study above and decided on the test problems for our experiments. We now design the reporting metrics to measure performance. Then we decide on the detailed experiment setup. We conduct some initial experiments with a base algorithm for this purpose.

3.3.1 Reporting Metrics

Our goal is to observe the behavior of run-times over changing problem sizes. But the target of our study is a stochastic search, which has no inherent state of being "finished".

The execution of an EA can be terminated at any point in time and earlier termination usually results in solutions of lower quality.

A Suitable Termination Condition

For practical use, EAs are usually equipped with one or more termination conditions. Examples by Eiben and Smith are the achievement of a predefined solution quality, the exhaustion of maximally allowed CPU time or fitness evaluations, the absence of major fitness improvement over a period of time, and the violation of a minimally required population diversity [28]. In our case, a termination condition that limits a time dependent variable is infeasible since those are our measurement goals. We decide to terminate an algorithm run

1. if a minimum solution quality is reached, which we consider as a successful run, or
2. if fitness of the best individual has converged, which we consider as an unsuccessful run.

We base the exact definition of fitness convergence on insight of initial experiments and discuss it in the next subsection. In order to find a suitable minimum fitness for the definition of success, we consulted results reported in literature.

Various authors use the *adjusted Rand index* [48] as a flexible quality measure to compare two clusterings. The index rates pairwise co-assignment of objects and is normalized to yield results around zero for random partitions and 1 for perfect agreement. In [25] and [39], results of search are compared to known solutions; in [78] results of different clustering techniques are compared against each other. Reported values of the index lie in the range of 0.9 to 1.0 for good results. We choose this range as a reasonable target quality.

We need to translate the target quality of the adjusted Rand index to a fitness value of the MMC function. For this, we experimentally investigate their correlation, based on the evaluation of systematically distorted reference solutions. Experiments are performed for 200 randomly generated clustering problems with $n = 500$, $k = 10$. We perturb the reference assignment by 1%, 2%, ... 20%, 25%, ... 50%, 60%, ... and 90%. On each level both the relative MMC fitness and the adjusted Rand index are measured. Figure 3.1 presents a scatter plot of the obtained results. Based on a target of 0.95 for the adjusted Rand index, we opt for a minimum MMC fitness of 85% of the reference fitness.

Measured Variables

An algorithm run terminates as soon as one of the termination conditions is fulfilled. Thus, not all runs terminate successfully. We decided to run repeated experiments for each problem size and report

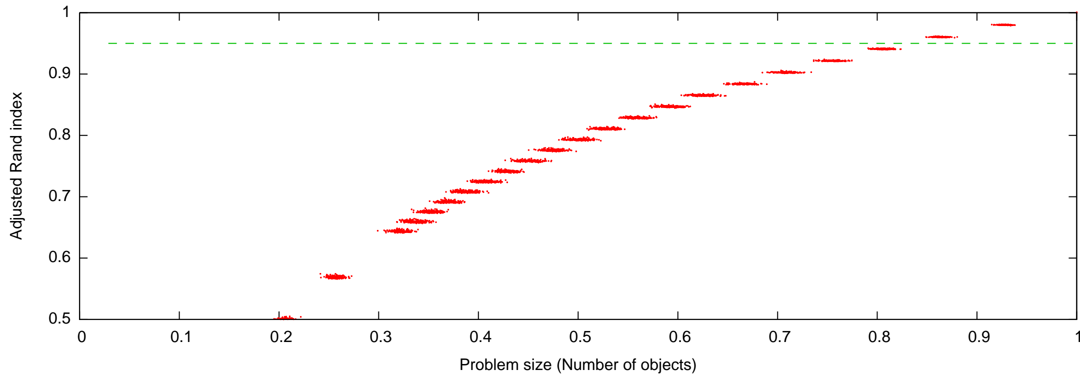


Figure 3.1: Scatter plot showing the correlation of the adjusted Rand index and MMC fitness, obtained through the evaluation of systematically distorted clustering solutions. A value of 0.95 is marked for the adjusted Rand index.

– the success rate (SR), which is the percentage of successful runs,

– the average success run-time (SRT), which is the mean run-time of the successful runs, and

– the expected run-time (ERT).

If an algorithm run terminates without success, one possibility is to restart the run with a different seed. This idea is the basis of ERT. To calculate ERT we measure the average run-time for unsuccessful runs (URT) as well. ERT can be defined recursively as

$$\mathrm{ERT} = \mathrm{SR} \cdot \mathrm{SRT} + (1 - \mathrm{SR})\,(\mathrm{URT} + \mathrm{ERT}) \ , \tag{3.1}$$

This equation can be expressed in a closed form as

$$\mathrm{ERT} = \mathrm{SRT} + \frac{1 - \mathrm{SR}}{\mathrm{SR}}\,\mathrm{URT} \ . \tag{3.2}$$

3.3.2 A Base Algorithm

We design a simple EA for clustering. The base algorithm is used in initial experiments to decide on some specifics of the experiment setup. It also serves as a reference point to compare improved EAs. The base algorithm uses a direct representation that encodes solutions as s string of length n, the number of objects to cluster. Each gene carries a value in the range $0 \ldots k-1$ to denote a cluster number. A population of 100 individuals is randomly initialized. We employ binary tournament selection and generational replacement with 1 elite individual. Reproduction is performed by uniform crossover with a

crossover rate of 1.0, i.e., it is always applied. The probability used in uniform crossover is set to 0.5, i.e., each parent has an equal chance to pass on its allele values. We use random resetting mutation, i.e., offspring are mutated by the reassignment of genes to a random cluster. We term the expected number of mutated genes the "mutation count" and set the per-gene mutation rate to $\frac{\text{mutation-count}}{\text{genome-length}}$. Initially we choose a mutation count of 1, i.e., each gene is mutated with a probability of $\frac{1}{\text{genome-length}}$. As decided in Section 3.2, the clustering criterion is MMC (Equation 2.1); fitness is calculated as the value of $\frac{1}{1+\text{MMC}}$ relative to a reference clustering, which needs to be maximized. We initially terminate a run if no relative fitness improvement of more than 1% can be observed for at least 500 consecutive generations.

3.3.3 Experimental Setup

The methodology we have defined so far leaves some open questions that we answer based on initial experiments. We use them to decide on a testing strategy, to validate the initially chosen parameters of the base algorithm, and to find a termination condition that reliably detects fitness convergence.

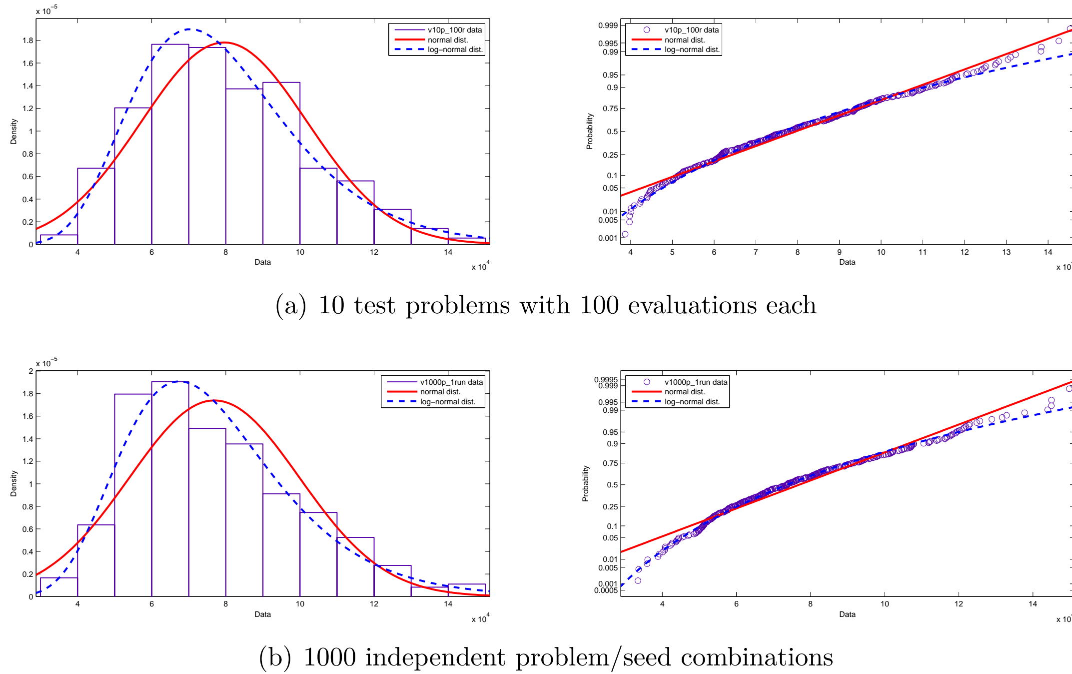
The Testing Strategy and a Statistical Analysis of Experiment Results

The experimental evaluation of an EA is influenced by two sources of randomness: EAs are stochastic search methods; the result of each run depends on the initial random number seed. On top of this, our test problems are randomly generated. Approaches in literature usually sample a few test problems and perform multiple runs on each; e.g., the authors of [39] run 10 problems 21 times each and average the combined results.

This might introduce unknown correlations, thus we suggest to randomly sample the space of all EA runs on all test problems. This is achieved by the independent random sampling of pairs of seeds. One seed is used to generate a test problem and the second to seed the EA run. We compare the suggested testing strategy with the traditional approach in literature by an initial experiment.

Figure 3.2 shows the run-times of successful runs out of 1,000 evaluations on 500-object, 10-cluster problems. Two test strategies are compared: *(i)* sampling 10 test problems and performing 100 EA runs on each, and *(ii)* sampling 1,000 EA runs out of the space of all runs on all problems. The success rates for both cases are similar; 35.7% for *(i)* and 36.2% for *(ii)*. The fitted probability distributions indicate a log-normal distribution of the results. We confirm this by Lilliefors' normality test [19], which rejects the normality hypothesis at the 5% significance level and cannot reject the log-normality hypothesis.

Assuming a log-normal distribution, we use a t-test [42] to compare both result sets. The hypothesis that both samples stem from the same distribution cannot be rejected at



(a) 10 test problems with 100 evaluations each

(b) 1000 independent problem/seed combinations

Figure 3.2: Histograms and probability plots for two different test configurations of problems with 500 objects and 10 clusters; the graphs show a fitted normal and log-normal distribution.

the 5% significance level. This confirms the validity of the suggested testing strategy. For the initial experiments above, we have used a total of 1,000 evaluated configurations. But to conduct further experiments in a feasible time, we decide to evaluate 200 randomly sampled pairs of a problem and an assigned seed. 200 pairs are generated once for each problem size. We then use the same combinations of problem and seed as input to all algorithm configurations.

Tuning the Base Algorithm Parameters

The parameters we have set for the base algorithm have been chosen intuitively, based on our previous experience with clustering EAs [24][37][83]. To assure their validity and explore the potential for improvement, we perform a small number of tests with different population sizes, mutation counts, and probabilities used in uniform crossover. The tests are run for a problem with 200 objects and 10 clusters, according to the test strategy defined above. We evaluate the following options:

1. Population size: 80, 100, 120

2. Mutation count: 0.5, 1, 2

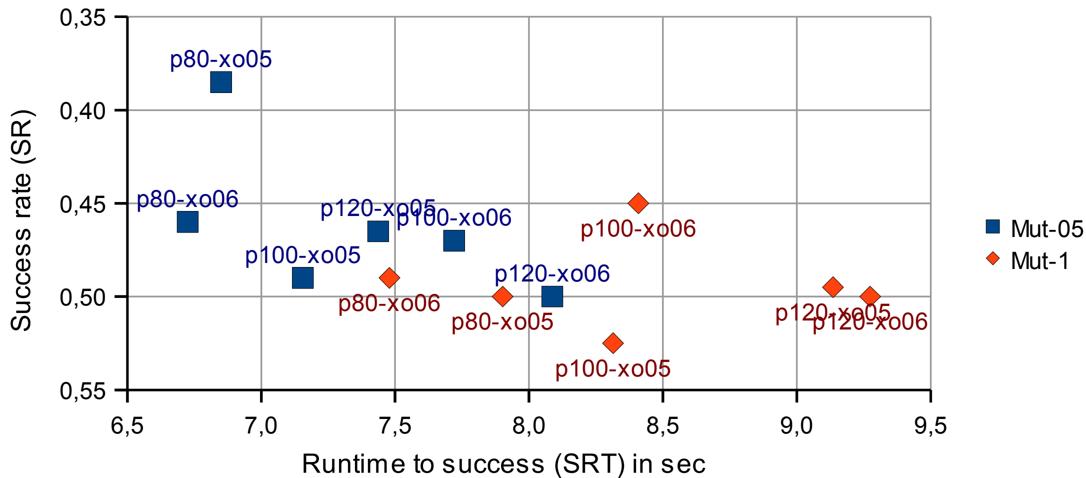


Figure 3.3: Scatter plot of SRT and SR for varying population size (pXX), selection probability in uniform crossover (xoYY) and mutation count (mut-ZZ).

3. Selection probability used in uniform crossover: 0.5, 0.6

Figure 3.3 presents the average SR and SRT found in experiments on 200-object, 10-cluster problems. Configurations with a mutation count of 2 are worse in both SRT and SR than the depicted configurations and lie outside the plotted region. We see that a smaller population size decreases SRT but comes at the risk of a lower SR. We decide to keep the initial population size of 100 as a good compromise. A lower mutation count is clearly beneficial when moving from 2 to 1. Lowering it further to 0.5 results in faster runtimes with not too much influence on the success rate. Thus we update the base algorithm to use a mutation count of 0.5. A clear trend cannot be seen for varying selection probability in uniform crossover. We keep the initial setting of the base algorithm, since a probability of 0.5, which gives equal chance to both parents, is most common in literature.

Detecting Fitness Convergence

In the initial experiments, we had defined fitness convergence in the base algorithm if no relative fitness improvement of more than 1% can be observed for at least 500 consecutive generations. To check the validity of this convergence criterion, we conduct initial experiments on large problems up to a size of 5,000 objects. This reveals a steep drop in success rates, from 43% for 500 objects, over 27% for 1,000 objects to only 6.5% for 5,000 objects.

We manually inspect the fitness plots of single runs. They show a decreasing slope for the fitness of the best individual with growing problem size. Thus the fixed time frame

of the initial convergence criterion fails to scale for large problems and terminates runs prematurely.

We perform a set of experiments with a relaxed termination condition that uses a time frame of 10,000 generations. Based on studying the resulting fitness plots we devise an updated termination condition: a run is considered as unsuccessful if the increase in relative fitness over the past 20% of generations falls below 1%; this termination condition is first tested once 125 generations have passed. This allows every run to continue for at least 125 generations. For example, in generation 125, the fitness levels of generations 100 and 125 are compared to check for the minimum relative fitness increase.

3.4 Summary

We summarize the experimental methodology, which we developed in this chapter:

- Test problems are artificially generated (Algorithm 3.1).
- For each problem category (number of objects and number of clusters), we fix 200 independent randomly sampled problem/seed pairs.
- Fitness is calculated as $\frac{1}{1+\text{MMC}}$ relative to a reference value.
- A fitness of 0.85 is considered a success.
- An unsuccessful run is terminated if the relative fitness improvement in the last 20% of the generations falls below 1%; the condition is tested on generations ≥ 125.
- The base algorithm uses a population size of 100, uniform crossover with a selection probability of 0.5, and mutates each gene with a $\frac{0.5}{\text{genome-length}}$ probability.

4 Improving Operators with Problem-specific Knowledge

We propose several ideas for introducing problem-specific knowledge into the operators of a simple evolutionary algorithm. Knowledge can be introduced in various steps of the general EA flow-chart, thus we categorize our proposals into modifications of initialization, crossover, and the introduction of hybridization approaches.

In this chapter we introduce our modifications, present the experimental setup we use to evaluate the modified algorithms, summarize the achieved results and conclude on the impact of these modifications on scalability.

4.1 Modifications to Initialization, Crossover and Hybridization

The base evolutionary algorithm, described in Section 3.3.2, uses general, out-of-the-box, operators in order to solve the clustering problem. Based on a direct encoding of a solution candidate as a string of cluster numbers, an initial population is randomly generated and then subjected to uniform crossover and a standard mutation. We propose modifications to this process, in order to implement problem specific knowledge in the algorithmic flow. We suggest two problem-specific initialization schemes in addition to random initialization, two alternative crossover operators, and two approaches on hybridizing the EA with improvement heuristics.

Across all categories we employ the concept of a "similarity heuristic", which reassigns objects in a solution candidate based on the summed similarity values of an object calculated per cluster. For a given clustering solution which assigns the objects into clusters $C_1, C_2, \dots, C_k$ we define

$$S_i(u) = \sum_{v \in C_i} s(u, v) \tag{4.1}$$

as the similarity sum of object u with respect to cluster i, using the pairwise similarity function $s(u, v)$. Algorithm 4.1 describes the similarity heuristic. It takes a set of objects

and a solution candidate as inputs and then reassigns the objects to clusters for which they have the highest similarity sum.

Algorithm 4.1: SimilarityHeuristic

```
1 begin
      Input: List<ObjectId> objList, int[ ] objAssignment
2     Randomly shuffle objList;
3     foreach u in objList do
4         Calculate S_i(u) for i = 1...k;
5         objAssignment[u] ← arg max_{i=1...k} S_i(u);
6 end
```

4.1.1 Alternative Initialization Schemes

In addition to random initialization, which creates a solution by assigning a cluster number that is uniformly drawn from $1 \dots k$ to each object, we introduce two alternative schemes. Based on the similarity heuristic introduced above we define the heuristic initialization, and based on minimum spanning trees (MST) we define the MST Initialization.

The Heuristic Initialization

The heuristic initialization is based on the random initialization with an additional post-processing of the random solution, which is subjected to one full run of the similarity heuristic. Thus all objects are reassigned, looking at them in a random order and placing each one into the cluster for which it achieves the highest similarity sum.

The MST Initialization

The MST initialization is based on a minimum spanning tree of the weighted graph $G = (V, E, w)$ where V, $|V| = n$, represents the objects to be clustered and each entry $s(u, v)$ in the sparse similarity matrix corresponds to an edge $e(u, v)$ with $w(e(u, v)) = 1 - s(u, v)$. In the clustering of graphs, MSTs have been used for a long time and are known to provide identical solutions to single-link clustering [36]. In the context of EAs Handl and Knowles have used MSTs for initialization: in their first paper that made use of MSTs, they repeatedly removed the longest edge from an MST to seed the initial population of a dynamic-k algorithm [38].

We use Kruskal's Algorithm to calculate an MST of G which yields $G' = (V, E')$ with $|E'| = n - 1$. To generate a k-clustering based on G', $k - 1$ edges have to be removed and

the remaining connected components induce the clustering. Instead of just selecting the $k-1$ least similar edges, we introduce two modifications that prevent the generation of singleton clusters and allow us to generate different k-cluster solutions.

We define an edge $e(u,v) \in E'$ as eligible for removal only if $\deg_{G'}(u) > 1$ and $\deg_{G'}(v) > 1$, which assures that the removal of e does not create a singleton cluster. Then we select $k-1$ edges randomly out of the $(k-1)+f$ least similar eligible edges, which are removed to induce a solution candidate. The parameter f introduces a certain freedom in choosing which edges should be removed and allows for the generation of multiple solution candidates based on a single MST.

4.1.2 Alternative Crossover Operators

The base EA uses uniform crossover to breed offspring based on two parent solution candidates. In combination with the employed direct encoding, this crossover method assigns each object the cluster number of either parent with equal chance. It is easily seen that this combination is not the best choice for clustering problems; e.g., two parents describing the same clustering but using permuted cluster numbers will create a differing offspring. To overcome this drawback we introduce two alternative crossover operators, the matching crossover and the cluster-based crossover.

The Matching Crossover

Matching crossover is an adaption of uniform crossover, which introduces an additional preprocessing step. Under the direct encoding each unique clustering solution has $k!$ possible genome representations; thus mixing the cluster numbers of two parents does not necessarily carry along any meaningful information. To compensate for this, we introduce a matching step that adjusts the cluster numbers in one parent to match those in the other parent. We aim to give "similar" clusters a higher chance of being labeled with the same cluster number in both parents. A similar approach has been used in [58] and [64], which adapted cluster numbers in the parents "in such a way that the difference between the two parent solutions is as small as possible."

To match cluster pairs we first count the number of co-assignments for all k^2 pairs of cluster numbers and then match those clusters which have the largest overlap. Algorithm 4.2 presents the employed preprocessing step in detail. After the matching process a regular uniform crossover is performed.

The Cluster-based Crossover

We introduce cluster-based crossover as a family of crossover operators following a common scheme that aims at preserving whole clusters during the crossover process. A

Algorithm 4.2: MatchNumbering

```
1  begin
       Input: int[ ] p1, int[ ] p2
2      coAssignment ← new int[k][k];
3      for i = 0 to n − 1 do                        /* Calculate co-assignment */
4          coAssignment [p1 [i]] [p2 [i]] + +;
5      match ← new int[k];
6      while not all clusters are matched do        /* Match clusters */
7          find the unmatched clusters i, j with maximal coAssignment[i][j];
8          match [i] ← j;
9      for i = 0 to n − 1 do
10         p1 [i] ← match [p1 [i]];
11 end
```

similar idea has been suggested by Falkenauer in [29]; he proposed to inject a subset of clusters of one parent unaltered into the other, resolving ambiguities in object assignment in favor of the newly injected clusters. This approach increases the total number of clusters and, depending on the problem, can create solution candidates that violate constraints. Falkenauer suggested to repair the resulting offspring through problem-dependent heuristics.

We decide on a slightly different approach by assigning different roles to the parents. Unlike uniform crossover or matching crossover, where both parents played an equal role, in cluster-based crossover one parent acts as the primary parent. Cluster-based crossover then creates one offspring from two parents in the following three steps:

1. In the first step, a number of clusters, say k_1, in the primary parent are selected for inheritance according to a certain selection strategy. The selected clusters are then copied to the offspring unaltered. The objects assigned in this step are excluded from any further processing.

2. In the second step, $k - k_1$ clusters of the secondary parent are selected, choosing those clusters that are still the most intact, i.e. those which have the highest percentage of yet unassigned objects. The yet unassigned objects in the selected clusters are added to the offspring, substituting their cluster numbers to prevent numbering collisions with the clusters selected in the first step.

3. After the first two steps a number of objects remains "homeless", namely those that belong in both parents to a cluster that has not been selected. The third and final crossover step assigns the homeless objects to a cluster according to a certain *homeless strategy*.

Algorithm 4.3: ClusterCrossover

```
1  begin
       Input: int[ ] p1, int[ ] p2
2      offspring ← new int[n], initialize with −1;
3      C^I ∈ 2^k ← selectPrimaryClusters(p1);
4      total^II, assigned^II ← new int[k];
5      for i = 0 to n − 1 do
6          if C^I_{p1[i]} = 1 then
7              offspring [i] ← p1 [i];
8              assigned^II [p2 [i]] + +;
9          total^II [p2 [i]] + +;
10     Set C^II ∈ 2^k selecting the clusters with minimal assigned^II / total^II values until
       |C^I| + |C^II| = k;
11     for i = 0 to n − 1 do
12         if C^II_{p2[i]} = 1 and offspring [i] = −1 then
13             offspring [i] ← subst(p2 [i]);
14     assignHomeless ({i : offspring [i] = −1});
15 end
```

Algorithm 4.3 describes the cluster-based crossover scheme in pseudo-code. Two steps in the crossover scheme allow for varying implementations, namely the selection strategy in the first step and the homeless strategy in the third step.

We suggest two alternatives for both steps, using either a random or an informed strategy for both cases. For the selection strategy we suggest to either

– randomly decide for each cluster in the primary parent whether it gets inherited to the offspring, giving equal probability to both cases. Thus the expected number of inherited clusters is $\frac{k}{2}$. Alternatively we

– aim to keep the best clusters in the primary parent by deterministically selecting the $\frac{k}{2}$ clusters which have the lowest value of inter-cluster similarity over intra-cluster similarity.

For the homeless strategy we suggest to either

– assign a random cluster number to the homeless objects, or

– run the similarity heuristic on the incomplete offspring, giving the homeless objects as the input.

4.1.3 Hybridization Approaches

As the last group of modifications we introduce two approaches to hybridize an EA with local search methods. We decide to invoke the introduced hybridization just before the regular mutation step happens, which allows the hybridized algorithm to keep its property of theoretically being able to mutate every solution candidate into every other solution candidate. Also some additional experiments, which we do not present here, showed no substantial difference when invoking the hybridization after the mutation. We introduce two approaches to hybridization, the similarity heuristic hybridization and the hill-climbing hybridization.

The Similarity Heuristic Hybridization

Our first hybridization executes the similarity heuristic, described earlier in Algorithm 4.1, for all newly generated solution candidates. The similarity heuristic hybridization processes all objects of the solution candidate in a random order, assigning them to the clusters for which they have the highest similarity sum.

The Hill-climbing Hybridization

Hybridizing an EA with the similarity heuristic gives no guarantee that the performed reassignments lead to an improvement in overall fitness. To compensate for this short-coming, we also introduce the computationally slightly more expensive hill-climbing hybridization. The general structure of the hill-climbing hybridization is very similar to the similarity heuristic hybridization, as it also reassigns all objects in a random order. But instead of selecting the new cluster of an object heuristically, the hill-climbing hybridization guarantees to perform the assignment that results in the highest fitness improvement.

To accomplish this, the change in fitness for each possible assignment of an object is calculated. Due to the nature of the min-max cut objective function, we do not need to completely recalculate the fitness for each assignment from scratch. The change in fitness can be quickly assessed, if the inter-cluster and intra-cluster similarity sums for all clusters are known and all $S_i(u)$ (Equation 4.1) values of the object under consideration have been calculated. For example, if object u is moved from cluster a to cluster b, the updated min-max cut (MMC') can be derived from the current one (MMC) as

$$\begin{aligned} MMC' = MMC &- \frac{\text{interClusterSum}(a)}{\text{intraClusterSum}(a)} + \frac{\text{interClusterSum}(a) + S_a(u)}{\text{intraClusterSum}(a) - S_a(u)} \\ &- \frac{\text{interClusterSum}(b)}{\text{intraClusterSum}(b)} + \frac{\text{interClusterSum}(b) - S_b(u)}{\text{intraClusterSum}(b) + S_b(u)} \end{aligned} \tag{4.2}$$

The main computational effort for both hybridizations is spent on calculating the similarity sums, $S_i(u)$, between each object and every cluster. Those values change with the reassignment of objects and can thus only be calculated as soon as an object is considered for reassignment. Computational complexity is easiest described using the graph interpretation of a problem as in Section 4.1.1: Calculating all $S_i(u)$ values for a single object needs $\mathcal{O}(\deg(u))$ time to consider all non-zero similarity values. When iterating over all objects each edge in the graph is involved exactly twice, once for each of its endpoints. This leads to a total complexity of $\mathcal{O}(2|E| + k|V|) = \mathcal{O}(|E| + k|V|)$ to calculate the similarity sums and to find the best out of k clusters for all $|V|$ nodes. In the similarity heuristic hybridization all involved operations are of an additive nature, the hill-climbing hybridization on the other hand requires an additional $\mathcal{O}(k|V|)$ floating point divisions to calculate Equation 4.2 for each object and each cluster.

4.2 Experimental Setup

Based on the operators presented above, we define configuration sets for initialization, crossover, and for the mutation and hybridization step. In this section we describe the defined configuration sets and present details about the machine we use to evaluate them.

All algorithm configurations for evaluation adhere to the general algorithm settings we have introduced in Section 3.3.3, using a generational EA with a population size of 100, binary tournament selection, a single elite individual, and a termination condition that stops an algorithm if the fitness of the best individual has not increased by more than 1% over the latest 20% of the elapsed generations.

4.2.1 System Setup

We evaluate all configurations on a machine with 4 dual-core, hyper-threading Intel Xeon CPUs running with 2.6 GHz that provides 2 GB of RAM to a 32 Bit Fedora 6 Linux operating system. Our evaluation setup runs 4 test instances in parallel, recording elapsed real-time, the average population fitness, and the fitness of the best individual for each generation. 200 problems are independently generated for each problem size and evaluated with common random numbers per problem for each algorithm configuration.

4.2.2 Evaluated Configurations

For initialization we decide to set up experiments with initial populations that are generated by different mixes of the proposed initialization methods. The MST initialization is configured with an f value of 5; for 10 cluster problems this gives $\binom{9+5}{9} = 2002$ combinations to sample initial individuals, enough to create a diverse population while only

Table 4.1: Algorithm configurations used for initialization

	Initialization type (% of individuals)		
	Random	Heuristic	MST
rand-init	100		
heur-init		100	
heur-rand-init	50	50	
mst-init			100
mst-rand-init	50		50
heur-mst-init		50	50
heur-mst-rand-init	33	34	33

Table 4.2: Algorithm configurations used for crossover

	Uniform	Matching	Cluster-based	
			Selection	Homeless strategy
uform-xo	✓			
match-xo		✓		
clust-xo			random	random
clust-xo-kb			keep-best	random
clust-xo-hh			random	heuristic
clust-xo-kb-hh			keep-best	heuristic

Table 4.3: Algorithm configurations used for mutation and hybridization

	Mutation count	Hybridization	
		Similarity Heuristic	Hill-climbing
mut05	0.5		
mut05-heur	0.5	✓	
mut05-hill	0.5		✓

considering the least similar edges for removal. Table 4.1 lists the set of initialization methods.

For crossover we elect to evaluate all configurations of the proposed operators, thus choosing uniform crossover, matching crossover, and the four possible strategy settings for cluster-based crossover. Table 4.2 lists the set of crossover operators.

For mutation we have not suggested any changed operators and always use a standard mutation with a mutation count of 0.5, meaning that each variable is mutated with a probability of $\frac{0.5}{\text{genome-length}}$. We have proposed two hybridizations which we execute before

the mutation step is performed. Accordingly, we evaluate three configurations, namely standard mutation used either alone or in combination with the similarity heuristic or hill-climbing. Table 4.3 lists the set of mutation and hybridization settings.

4.3 Experimental Results

The 7 initialization schemes, 6 crossover configurations, and 3 mutation configurations described above result in a total of 126 different configurations for the evolutionary algorithm. We initially assessed all configurations on 10-cluster problems consisting of 100, 500, 1000, and 2000 objects. The complete measurement results along are presented in Appendix A.

We find that crossover and hybridization are the main factors in achieving high success rates; three configuration settings in these areas constantly result in near-perfect ($\geq 97\%$) success:

- The hill-climbing hybridization, regardless of the chosen crossover or initialization,
- Cluster-based crossover with a random selection strategy and a heuristic homeless strategy, regardless of the chosen hybridization or initialization, and
- Cluster-based crossover with a random selection strategy and a random homeless strategy in combination with the similarity heuristic hybridization, regardless of the chosen initialization.

Algorithm configurations with other crossover and hybridization settings show greatly varying results in success, sometimes exhibiting near-constant failure. To allow for a further classification of the algorithm configurations, we look in detail on the fitness development during an algorithm run. Based on the development of the fitness of the best individual and the average population fitness in the algorithm runs, we classify the observed results into families of algorithm configurations, shown in Table 4.4.

In the area of initialization settings we find the heuristic initialization to be dominant. All initialization mixes containing heuristically initialized individuals exhibit a behavior similar to a pure heuristic initialization. The MST initialization on the other hand shows a varying dominance. Mixing MST initialized individuals with randomly initialized individuals results sometimes in a behavior similar to a purely MST initialized population and in other cases similar to a purely randomly initialized one.

The following subsections present the classes we found in more detail and show exemplary fitness plots for each class. We first analyze the impact of the similarity heuristic hybridization in those settings that do not show near-perfect success, then we look at the results found for various combinations of crossover and initialization in a non-hybridized

Table 4.4: Families of algorithm configurations exhibiting similar behavior, * is a placeholder for all settings in a category, heur* stands for all initializations that include heuristic initialization

Class	Crossover	Initialization	Hybridization
S1	*	*	mut05-hill
S2	clust-xo, clust-xo-hh	*	mut05-heur
S3	clust-xo-hh	*	mut05
H1	clust-xo-kb, clust-xo-kb-hh, match-xo	heur*, rand	mut05-heur
H2	clust-xo-kb, clust-xo-kb-hh, match-xo, uform-xo	mst, mst-rand	mut05-heur
H3	uform-xo	heur*, rand	mut05-heur
M1	uform-xo, match-xo	heur*, rand, mst-rand	mut05
M2	uform-xo	mst	mut05
M3	match-xo	mst	mut05
M4	clust-xo	*	mut05
M5	clust-xo-kb	heur*, rand	mut05
M6	clust-xo-kb-hh	heur*, rand	mut05
M7	clust-xo-kb, clust-xo-kb-hh	mst, mst-rand	mut05

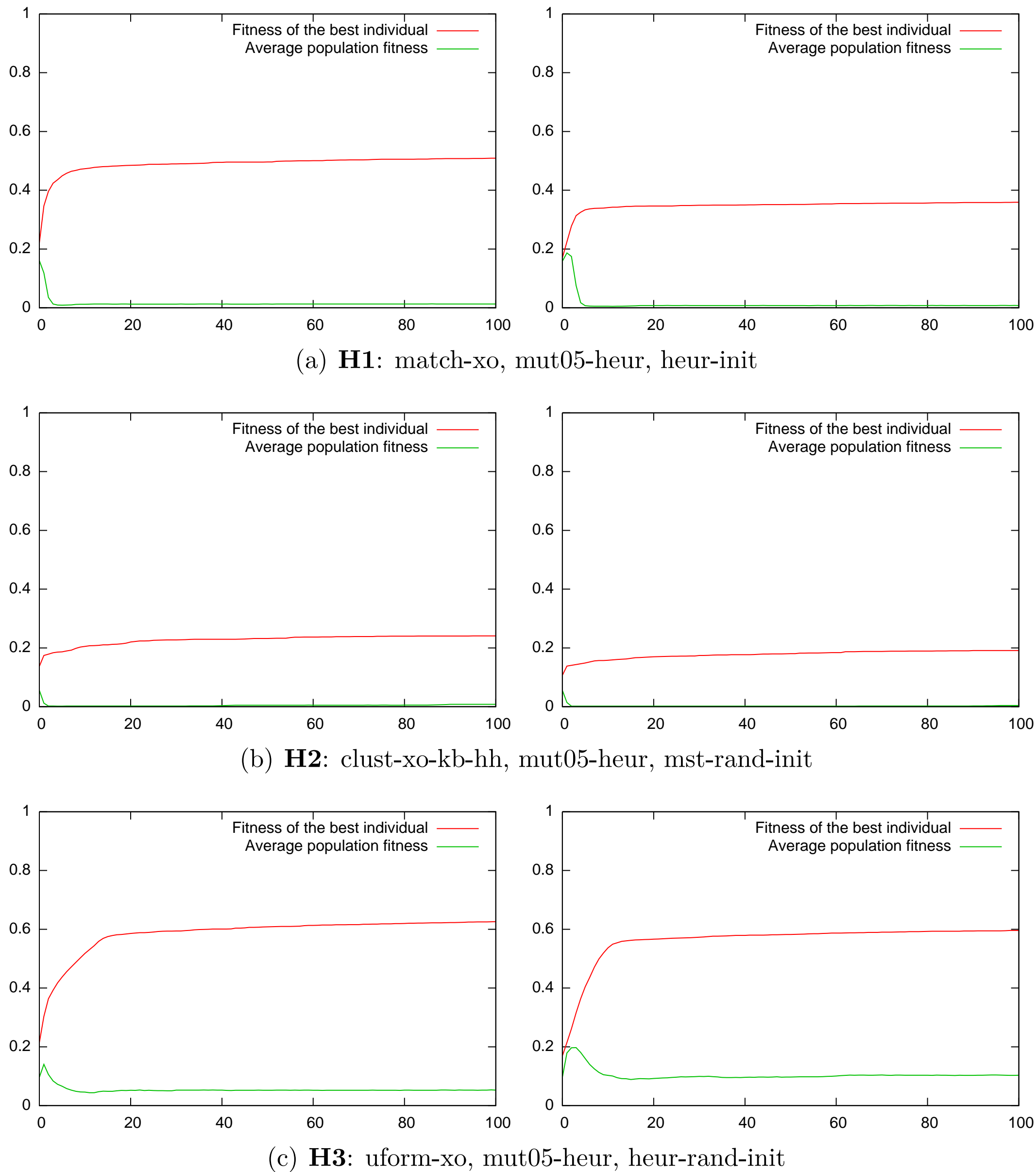
setting, and lastly we discuss the findings on the configurations that constantly show near-perfect success.

4.3.1 The Similarity Heuristic Hybridization in Unsuccessful Runs

Algorithm configurations which employ the similarity heuristic and do not exhibit near-perfect success fall into one of three classes **H1** to **H3**, depending on crossover and initialization. Figure 4.1 shows exemplary fitness plots for each class. We find that configurations that have not been initialized with the MST initialization or a combination thereof with random initialization (classes **H1**, **H3**) show an initial increase in fitness of the best individual, whose value at least doubles and then converges until generation 20. Configurations using the MST or MST/Random initialization (class **H2**) show little to no improvement in the fitness of the best individual.

We further observe that the average population fitness for nearly all configurations (classes **H1**, **H2**) drops to a value close to zero, doing so in an instant for MST or MST/Random initialized populations and over a course of at most 5 generations for other initializations. The only exception is class **H3**, comprised of configurations that use uniform crossover with any initialization but MST or MST/Random. Those algorithm configurations show an average fitness that is distinctly different from zero.

We hypothesize that the observed behavior has two reasons: First, we suspect the heuristic hybridization to produce a lot of degenerate solution candidates, clusters that contain at least one empty cluster and thus have a fitness value of zero. Since the heuristic solely considers the summed similarity of an object towards each cluster, a



(a) **H1**: match-xo, mut05-heur, heur-init

(b) **H2**: clust-xo-kb-hh, mut05-heur, mst-rand-init

(c) **H3**: uform-xo, mut05-heur, heur-rand-init

Figure 4.1: Exemplary fitness plots for classes **H1** to **H3**. The y-axis shows fitness values, averaged over 200 runs; the x-axis shows the number of generations. Graphs on the left show the 500-object problem, those on the right show the 2,000-object problem.

cluster with many objects of little similarity can seem more favorable than a smaller cluster with objects of higher similarity. Thus a solution candidate with an imbalance in cluster sizes might grow its largest cluster up to the point where one of the smaller clusters looses all of its members.

If degenerate solutions are selected for crossover, the informed crossover operators might work to preserve it; e.g., Algorithm 4.2 of the matching crossover will pair two de-

Table 4.5: Percentage of degenerate individuals in the population, tracked over 10 generations of the match-xo, mut05-heur, heur-init algorithm (averages of 200 runs)

Problem size	Generation									
	0	1	2	3	4	5	6	7	8	9
100	0.85	0.99	0.99	0.99	0.99	0.99	0.99	0.99	0.99	0.99
500	0	0.09	0.57	0.87	0.94	0.95	0.96	0.95	0.95	0.95
1000	0	0	0.08	0.57	0.87	0.94	0.95	0.95	0.95	0.95
2000	0	0	0	0.12	0.66	0.91	0.95	0.95	0.96	0.96

generate clusters and thus prevents an offspring from recovering. Only uniform crossover has a high chance of populating all clusters during crossover, since it is unlikely that two identically numbered clusters are both degenerate, if only a few degenerate clusters exist. Accordingly class **H3** shows a non-zero average population fitness.

Our second hypothesis relates to the particularly disastrous failure of the MST initialization. If the assumptions about the similarity heuristic hybridization are true, then an initialization scheme that produces a high imbalance in cluster sizes would reinforce the tendency towards failure. Thus we suspect the MST initialization to produce such solution candidates, which would explain the instant drop in average fitness.

To check the likelihood of our hypotheses we collected additional statistics for two exemplary algorithm configurations. Table 4.5 shows the number of degenerate solution candidates during the first 10 generations of an algorithm configured with match-xo, mut05-heur, and heur-init. The results clearly show that nearly all solution candidates in the population consist of degenerate individuals as we have assumed. We ran the same algorithm with a modified similarity heuristic, in which we divide the summed similarity between an object and a cluster by the number of edges that connect that object to the cluster. An object is then assigned to the cluster with the highest average similarity. This should remove the effect that a cluster with many objects of little similarity is too attractive. We find that with the changed similarity heuristic, the percentage of degenerate clusters decreased drastically: e.g., for the 100-object problems, the algorithm configuration of Table 4.5 resulted in less then 5% degenerate individuals in generation 9. But if run for a greater number of generations, the algorithm with the updated similarity heuristic always converged at a low fitness level and never found a satisfying solution. The improve percentage of degenerate individuals suggests that further research on updates to the similarity heuristic might be worthwhile. But we could not perform those in the scope of this work.

To check our second assumption, we collected statistics of the MST initialization, to get insights on the probability that the removal of an MST-edge creates a very small cluster of just a couple of nodes. Such clusters would be especially susceptible to degeneration

and a high occurrence of them will explain the observed behavior in class **H2**. Averaging over the MST-initialized individuals of all 200 problems in each size category, we counted the number of very small clusters. We find that for the 100-object problem 3.2 clusters contain 3 or less objects, for the 500-object problem 6.8, for the 1000-object problem 8.2, and for the 2000-object problem 8.5 clusters contain 3 or less objects. This confirms our assumption about the MST initialization.

4.3.2 Modified Crossover and Initialization without Hybridization

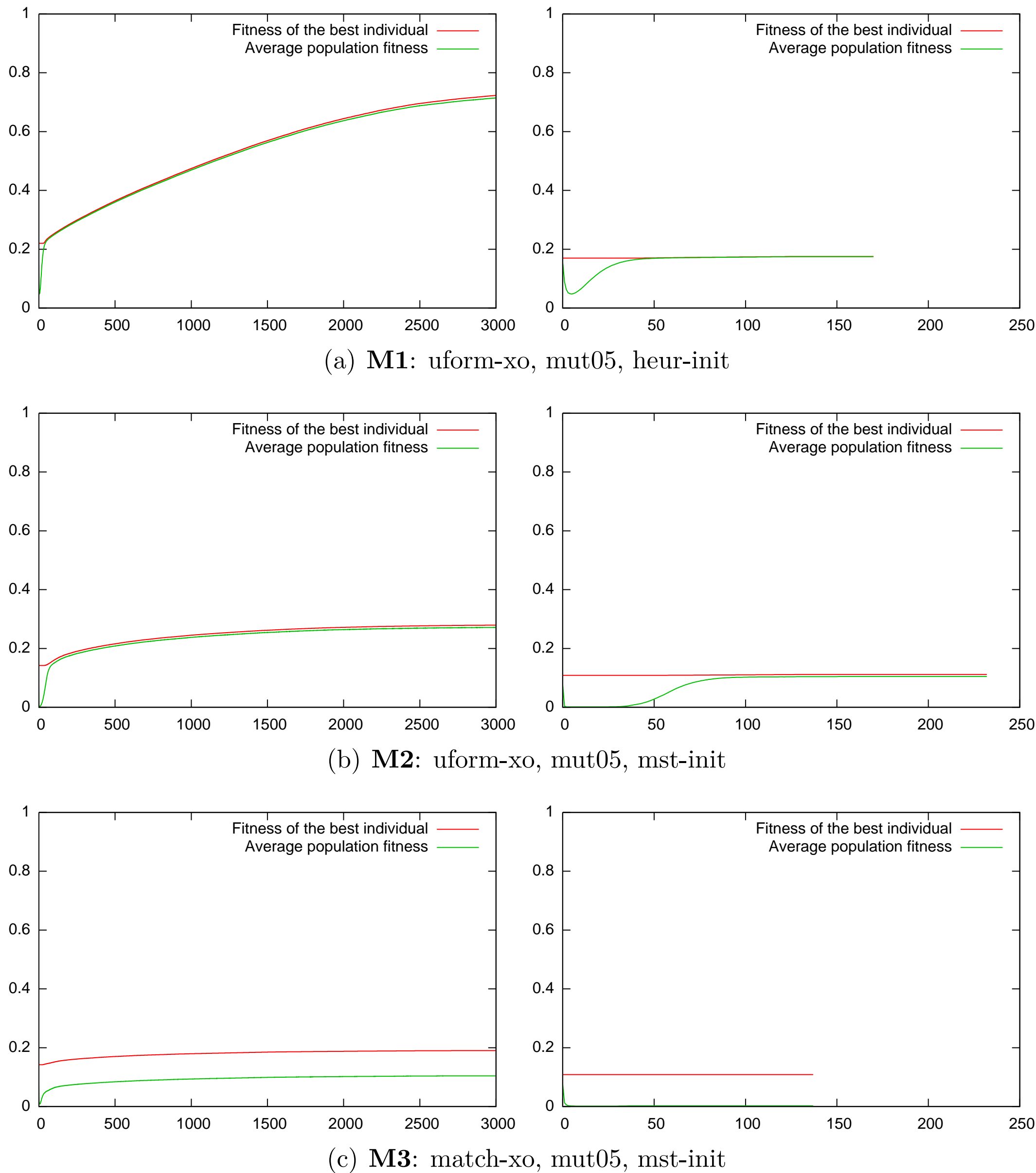
In the case where crossover and initialization are not hybridized with a local search, only the cluster-based crossover with a random selection and a heuristic homeless strategy shows near-perfect success. Other crossover operators show varied success, which in most cases depends on the chosen initialization scheme as well.

The following subsections describe the classes of algorithm configuration **M1** to **M7**. We first look at those cases where uniform or matching crossover is applied, then on those using variations of cluster-based crossover.

Uniform and Matching Crossover

Results for algorithm configurations that use uniform or matching crossover are shown in Figure 4.2 as exemplary fitness plots. In all combinations, where initialization is not purely MST based, the algorithm configurations fall into class **M1**. Similar to all instances we find that, independent of the configuration details, the average population fitness closely follows the fitness of the best individual on all runs. Both values increase rather slowly, which results in rather high average run-times to success.

We see the cause for the observed behavior in a reduction of the EA to a pure local search after some generations. On a detailed inspection of some configurations we found that, using uniform crossover, after 250 generations nearly all runs show genetic convergence, meaning over 95% of the genes have the same allele value on over 95% of the individuals. The required generations to see the same effect for matching crossover is only slightly higher (Table 4.6). The required generations to convergence generally drop with growing genome size; this is most likely rooted in the smaller mutation rate for the larger problems, since each gene is mutated with a probability of $\frac{0.5}{\text{genome size}}$. Once genetic convergence has occurred, matching crossover behaves exactly like uniform crossover; if both parents largely agree in their clusters and assigned cluster numbers the renumbering algorithm has no effect. Crossover can then only introduce small changes to an individual, which act in unison with mutation like a parallel execution of multiple local searches. This local search process is responsible for most of the time spent in such algorithms, since the generations required until success reach to the thousands.



(a) **M1**: uform-xo, mut05, heur-init

(b) **M2**: uform-xo, mut05, mst-init

(c) **M3**: match-xo, mut05, mst-init

Figure 4.2: Exemplary fitness plots for classes **M1** to **M3**. The y-axis shows fitness values, averaged over 200 runs; the x-axis shows the number of generations. Graphs on the left show the 500-object problem, those on the right show the 2,000-object problem. Fitness increases too slow to prevent premature termination for large problems with an intelligent initialization.

The usage of initialization methods that are not purely MST based have two effects. First of all, an intelligent initialization starts the algorithm run with a higher fitness of the best individual. This can lead to premature termination of a run, since the termination condition asks for a minimum relative increase in this value over a time-frame that

Table 4.6: Generations to genetic convergence for the mut05, rand-init algorithm using uform-xo and matching-xo (based on 200 runs). Q_3 denotes the upper quartile.

Problem size	Uniform Crossover			Matching Crossover		
	Median	Q_3	Maximum	Median	Q_3	Maximum
100	162	184	253	245	293	486
500	197	214	284	212	234	322
1000	176	187	242	188	202	244
2000	171	178	198	178	186	322

depends on the number of elapsed generations. This effect can be seen especially in the 2000-object problem, which nearly never reaches success if an intelligent initialization is used.

To allow an analysis of the effect of intelligent initialization, in spite of premature termination, we look at the cleaned success rates of the 500 and 1000-object problems. The termination condition is first checked after 125 generations have elapsed. A couple of runs that use intelligent initialization are terminated at this point or the closely following generations. Thus we remove all runs that terminate before generation 200 and calculate the cleaned success rates shown in Table 4.7, which are based on only those runs that terminate at a later generation. We still see lower success rates compared to random initialization. Together with the measured success run-times this suggests that algorithm runs starting from an intelligent initialization reach their local optimum faster, but usually converge at a lower quality optimum. We did not show details for MST/random initialization, since its initial fitness increase is so slow that premature termination generally happens. Tests with a relaxed termination condition suggested that it behaves along the lines of the other presented initialization schemes and shows early convergence at a low fitness value.

Configurations that combine uniform or matching crossover with a pure MST initialization show a behavior different from the initializations discussed above. Since most MST-initialized runs are terminated very early, we have again tested the configuration with a relaxed termination condition. We find that common to both crossovers is an initial decline in the average population fitness to almost zero; a behavior that fits to the susceptibility of MST-initialized individuals to yield solutions that quickly result in degenerate clusters, as we found in combination with the similarity heuristic. In combination with uniform crossover (class **M2**) the average population fitness manages to recover after a while and approximates the value of the best individual. But further on, the run fails to reach a local optimum of any notable quality through the subsequent local search.

Table 4.7: Cleaned success rates for uniform or matching crossover with various initialization methods

Initialization	Uniform Crossover				Matching Crossover			
	500 objects		1000 objects		500 objects		1000 objects	
	Runs	SR	Runs	SR	Runs	SR	Runs	SR
rand-init	200	0.33	200	0.41	200	0.40	200	0.38
heur-init	200	0.28	163	0.29	199	0.24	162	0.33
heur-mst5-init	196	0.26	158	0.37	198	0.28	168	0.29
heur-mst5-rand-init	195	0.22	168	0.26	198	0.22	161	0.26
heur-rand-init	199	0.25	164	0.32	196	0.26	154	0.34

In the case of matching crossover (class **M3**) the average fitness does not recover for problem sizes of 1000 or more objects. We observed some algorithm runs in detail and found that most individuals contain small clusters that are not degenerate in size but consist only of objects with a mutual similarity of zero; a combination that also leads to a zero MMC fitness. This phenomenon is most likely caused by situations where non-overlapping clusters of just a few objects are matched during crossover and thus cause unrelated objects to end up in the same cluster.

Cluster-based Crossover

The results seen from algorithms that employ cluster-based crossover without any hybridization is greatly dependent on the chosen strategies for cluster selection and homeless assignment. If both tasks are done randomly, the resulting configuration falls into class **M4**, which is shown in Figure 4.3. General fitness behavior is similar to uniform or matching crossover: the average population fitness approximates the fitness of the best individual and both start to improve closely together. But the improvement rate for cluster-based crossover is much lower than what we see with uniform or matching crossover. This explains the lack of success for algorithm configurations in this class, since the fitness increase is too slow to prevent the termination of an algorithm run.

In additional experiments on the 1000-object problems, using a relaxed termination condition, we find that for cluster-based crossover with heuristic initialization the average fitness of 200 runs at generation 10,000 is 0.25; uniform crossover with random initialization, on the other hand, reaches an average fitness of 0.75 in the same number of generations. A likely reason for the observed behavior is the effect of the random assignment of homeless nodes. By chance most homeless nodes will be assigned to the "wrong" cluster during this step. This assignment is then later on preserved for all clus-

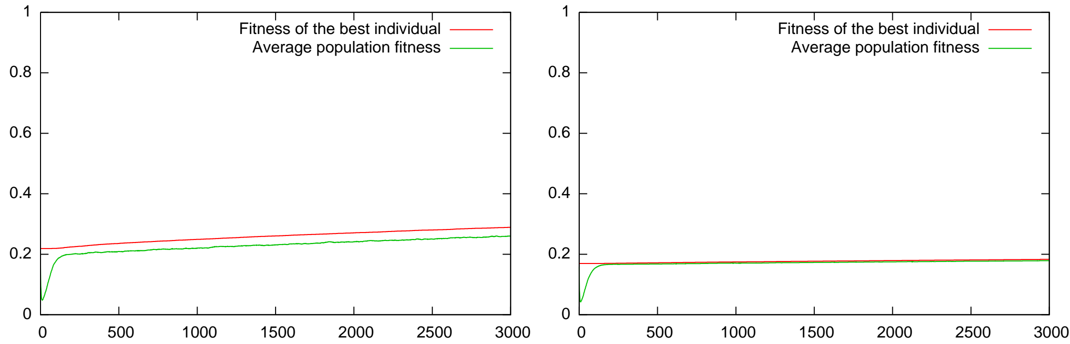


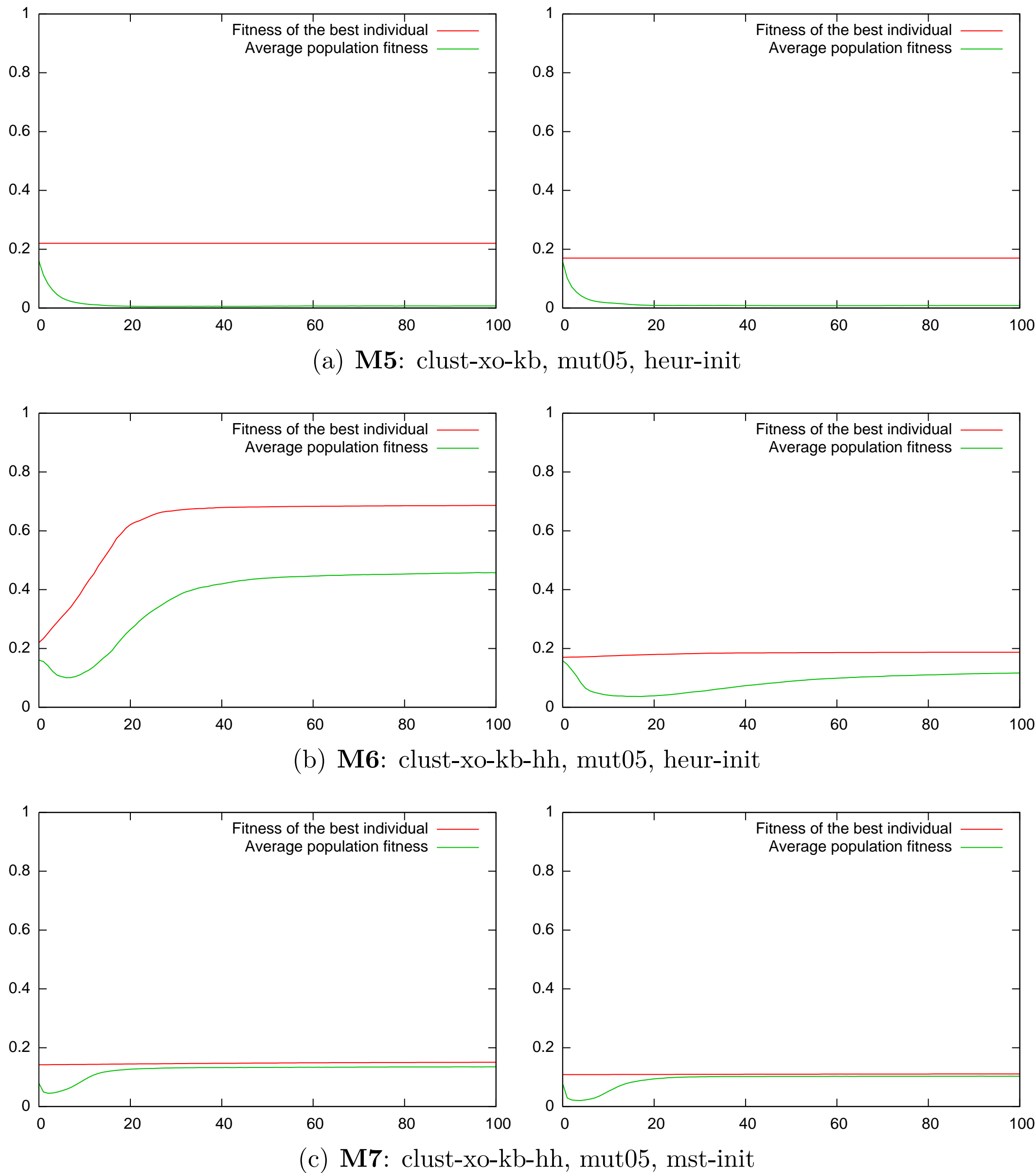
Figure 4.3: Exemplary fitness plots for class **M4**. The figures plot fitness values vs. the number of generations. The graph on the left shows the 500-object problem, the one on the right shows the 2,000-object problem. The plots are averaged over 200 runs of the clust-xo, mut05, heur-init algorithm with the standard termination condition disabled.

ters that are selected in a primary parent; this might drastically limit the possibility to generate good offspring during crossover.

Unlike other crossovers, class **M4** also contains the purely MST initialized configuration. This is due to the possibility to reject the "large cluster" in both parents during the first breeding round. If this happens, nearly all objects end up homeless and are randomly assigned. Thus this configuration closely resembles a random initialization.

Algorithm configurations of cluster-based crossover with a keep best selection strategy show a more varied behavior. Class **M5** contains those configurations that use a random homeless strategy combined with any initialization but MST or MST/Random initialization. Exemplary fitness plots are shown in Figure 4.4(a). Common to those configurations is a drop of the average population fitness to nearly zero in about 20 generations, while the fitness of the best individual does not improve. A detailed look at the best-cluster selection shows that usually the largest clusters in the primary parent are kept; a behavior that matches the characteristics of the min-max cut, which reaches its overall optimum for a single large cluster. Due to the random assignment of homeless objects, each cluster receives about $\frac{1}{k}$-th of them, which leads to the further growth of large clusters. Thus over time some large clusters keep growing, while the remaining clusters loose most objects. The small clusters suffer a similar fate as those in class **M3** and thus the fitness of most individuals drops to zero.

If cluster-based crossover is used with a keep best selection strategy and a heuristic homeless strategy it shows a different behavior. Class **M6** contains those configurations, again for all initializations but MST or MST/Random initialization. Exemplary fitness plots are shown in Figure 4.4(b). Unlike the random homeless strategy, the algorithm configurations in this class show an increase in the average population fitness and also improve on the fitness of the best individual. This shows that the heuristic homeless



(a) **M5**: clust-xo-kb, mut05, heur-init

(b) **M6**: clust-xo-kb-hh, mut05, heur-init

(c) **M7**: clust-xo-kb-hh, mut05, mst-init

Figure 4.4: Exemplary fitness plots for classes **M5** to **M7**. The y-axis shows fitness values, averaged over 200 runs; the x-axis shows the number of generations. Graphs on the left show the 500-object problem, those on the right show the 2,000-object problem.

strategy allows for more appropriate assignments, it prevents the large clusters from taking in unfitting objects. Applying the similarity heuristic only to few objects prevents the drawback described in 4.3.1 of creating degenerate clusters. Still, the larger the problem size gets, the smaller the achieved improvement in this class. We suspect this is caused by a limited capability to explore the search space, since the deterministic

selection tends to choose the large clusters of the primary parent, which leads to them always being kept intact.

Class **M7** contains algorithm configurations using cluster-based crossover with a keep best selection strategy in combination with a MST or MST/Random based population initialization, regardless of the chosen homeless strategy. Exemplary fitness plots are shown in Figure 4.4(c). We observe that these configurations are generally not able to improve the fitness of the best individual by a significant amount, while the average population fitness approximates the fitness of the best individual. If pure MST initialization takes place, the average fitness converges quickly, for combined initialization this process takes considerably longer and is usually interrupted by the termination condition. Upon a detailed inspection of several algorithm runs, we find that the single large cluster of MST-initialized individuals has the most favorable MMC addend and is thus always inherited completely if an MST-initialized individual is chosen as the primary parent. This allows the oversized cluster to spread to the whole population, which then converges at a low local optimum.

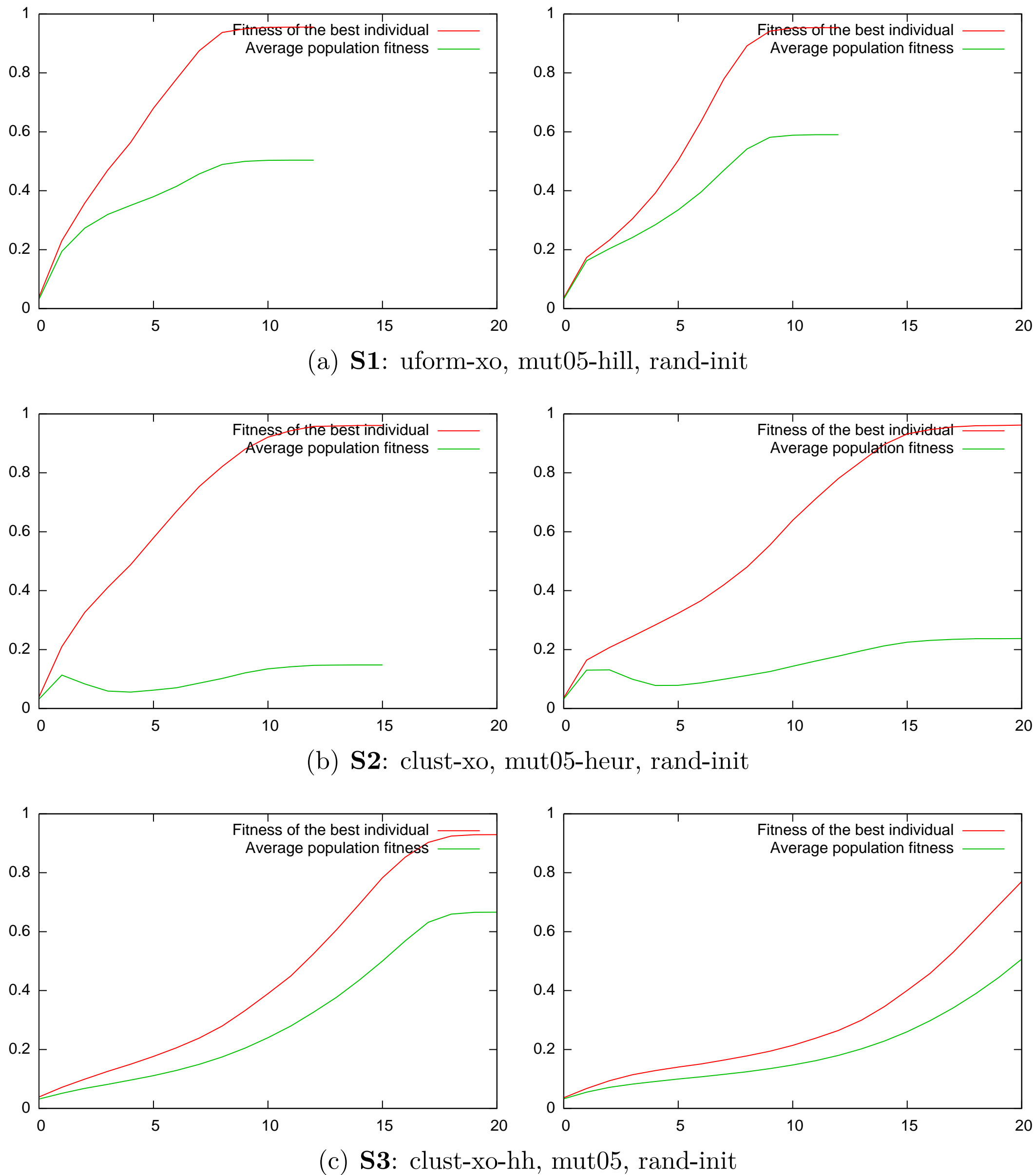
4.3.3 Configurations Resulting in Near-Perfect Success

After having looked in detail at those algorithm configurations that fail to reach a satisfying solution quality reliably, we now turn to the configurations that do. We find that in those configurations, hybridization is the most dominant factor that affects algorithm behavior. We first describe the characteristics of those algorithms employing the hill-climbing heuristic, then present the results achieved with cluster-based crossover in combination with the similarity heuristic hybridization and finally discuss the non-hybridized cluster-based crossover.

The Hill-climbing Heuristic

The largest group of successful algorithms is made up of configurations that use the hill-climbing hybridization (class **S1**). This results in constant success, regardless of the selected crossover or initialization scheme. We find that these algorithms find a satisfying solution quickly, requiring no more than 15 generations to do so. Exemplary fitness plots are shown in Figure 4.5(a).

To see the unbiased effect the involved hill climber has on performance, we conducted some iterative local search experiments. The 200 problem instances of each problem size are iteratively processed by the hill-climbing hybridization, starting from a single initial solution candidate. Table 4.8 presents the achieved success rates and the average required iterations to success, when repeating the hill-climbing for at most 500 iterations.



(a) **S1**: uform-xo, mut05-hill, rand-init

(b) **S2**: clust-xo, mut05-heur, rand-init

(c) **S3**: clust-xo-hh, mut05, rand-init

Figure 4.5: Exemplary fitness plots for classes **S1** to **S3**. The y-axis shows fitness values, averaged over 200 runs; the x-axis shows the number of generations. Graphs on the left show the 500-object problem, those on the right show the 2,000-object problem.

We find that only the smallest problem size had a high probability of being trapped in a local optimum, with success rates below 50% and even zero success on heuristically initialized individuals. The latter is most likely caused by initial individuals which contain a degenerate cluster, a condition that is likely to appear for small problems (cf. Table 4.5). For the larger problem sizes, satisfying success rates of around 80% are achieved.

Table 4.8: Success rates and average iterations till success for iterative hill-climbing on a single solution candidate, based on the hill-climbing hybridization

Initialization	100		500		1000		2000	
	SR	Iter.	SR	Iter.	SR	Iter.	SR	Iter.
rand-init	0.44	4.9	0.89	8.6	0.82	11.1	0.86	13.3
heur-init			0.88	6.6	0.85	9.6	0.87	12.5
mst5-init	0.30	2.9	0.69	6.6	0.70	8.4	0.76	9.8

This renders our EAs that employ the hill-climbing heuristic similar to the approaches presented in Section 2.4.2, which combine k-Means with an EA. Thus the main focus of the EA lies no longer on the performed crossover and mutation, but on compensating the shortcoming of a heuristic that might result in local optima of undesired quality for some initial solution candidates.

Cluster-based Crossover with the Similarity Heuristic Hybridization

Earlier we saw that some algorithm configurations that use the similarity heuristic hybridization nearly never reach successful results. We found the reason for this in the tendency of the heuristic towards growing large clusters. If used in conjunction with cluster-based crossover, configured with a random selection strategy (class **S2**), the algorithm is able to counter those effects and achieve satisfying results. Exemplary fitness plots are shown in Figure 4.5(b).

The root for this change in behavior is most likely the possibility of cluster-based crossover to eliminate large clusters in those cases when they are not chosen to be kept in the primary parent. This can prevent solution candidates from degenerating and thus allows the algorithm to reach higher quality solutions. But looking at the average population fitness of this class, we find that it shows only a low fitness and no clear improvement during the run. This indicates that most solution candidates are still degenerate and only a few reach good results.

Non-hybridized Cluster-based Crossover

Algorithm configurations in class **S3** use cluster-based crossover with a random selection strategy and a heuristic homeless strategy, without any hybridization. The fitness plots in Figure 4.5(c) show that in this case not only the fitness of the best individual, but also the average population fitness exhibit a good improvement. This shows that this crossover configuration has the chance to transfer beneficial properties of the parents to offspring: heuristically assigning homeless nodes results in appropriate clusters, which

allows a fitness growth. The abandonment of using the similarity heuristic on all objects prevents degeneration of solution candidates, which impaired the average fitness in the previous class.

Although the required generations until success are more than for the hybridized approaches, the achieved run times are still in line with those found for the algorithms using hill-climbing hybridization. This shows that the time spent per generation is less in the non-hybridized case; which is in line with our expectations, since we do not have to pay the hybridization costs. We will compare the achieved run-times in depth in the following section.

4.4 The Impact on Scalability

In the previous sections of this chapter we have presented various ideas for introducing problem-specific knowledge into an evolutionary algorithm for clustering. In extensive experiments we found that our suggestions can easily miss the target of improving performance, if they are used in a bad combination. Many of the analyzed algorithm configurations resulted in a serious degradation of success, which we could explain in most cases by the interaction between the used genetic operators.

We will not look at those unsuccessful configurations in any greater detail, but focus our attention on the remaining cases. Here we find two large groups that showed mostly consistent results: First, non-hybridized algorithms that keep the initial uniform crossover or use the closely related matching crossover. They exhibited run-times and success rates similar to the initial simple EA. Second, those algorithm configurations that constantly result in near-perfect success.

In the remainder of this section, we will first discuss the suitability of minor changes, such as introducing only improved initialization, to improve scalability. Second, we will have an in-depth look at the scalability of the very successful configurations and analyze how they behave for even larger problem sizes.

4.4.1 Using Intelligent Initialization with Traditional Crossover

The base algorithm shows unfavorable performance for problems of up to 2,000 objects. Fitting a power function to SRT over problem size ($\mathrm{SRT} = a \cdot \mathrm{size}^b + c$) results in an exponent of $b = 2.41$ with a 95% confidence interval of $(2.19, 2.63)$. This strongly indicates worse than quadratic scalability; the average SRT for a 2,000-object problem takes already longer than 20 minutes. Thus the basic algorithm is unsuitable for large problems.

The other algorithms of class **M1** in Section 4.3.2 also show only a slow fitness improvement; success rates range from 25% to 45%. Using intelligent initialization reduces

Table 4.9: Average generations and run-time to success, expected run-time for algorithms using uniform crossover or matching crossover with intelligent initialization. Runs that are faster than random initialization are printed in bold.

Uniform Crossover	100			500			1,000		
Initialization	Gen.	SRT	ERT	Gen.	SRT	ERT	Gen.	SRT	ERT
rand-init	310	1.40	2.93	3683	50.62	189.69	9141	251.28	743.40
heur-init	204	1.09	**2.61**	2724	38.39	**172.49**	6852	190.32	819.01
heur-mst-init	100	0.75	**2.87**	2758	39.30	190.32	7870	215.79	**675.11**
heur-mst-rand-init	104	0.76	3.93	2570	36.18	236.36	7414	204.64	1003.49
heur-rand-init	210	1.13	2.96	2793	39.64	208.57	7023	193.43	749.25

Matching Crossover	100			500			1,000		
Initialization	Gen.	SRT	ERT	Gen.	SRT	ERT	Gen.	SRT	ERT
rand-init	373	1.99	4.41	3506	52.11	155.99	9112	259.86	834.90
heur-init	196	1.33	4.57	2807	42.36	216.21	6840	198.67	**766.13**
heur-mst-init	69	0.76	**3.68**	2736	42.09	181.98	7373	211.66	903.39
heur-mst-rand-init	74	0.80	5.42	2800	42.26	237.49	7668	220.91	1018.88
heur-rand-init	262	1.62	**4.30**	2712	41.40	210.06	7543	216.84	**778.44**

the time required until convergence, but also reduces the average fitness level that the algorithm has reached upon convergence, thus resulting in lower success rates. Replacing uniform crossover with the similar matching crossover has a more mixed effect; for varying initialization schemes and problem sizes both increase and decrease of success rates and the times required to convergence can be observed.

To research the potential of the modified configurations for improved performance, we need to weigh the benefits of decreased run-time against the drawbacks of the simultaneously decreasing success rate. For this reason we will look at the expected run-time (Equation 3.2), which combines both factors.

In Table 4.9 we have compiled the average number of generations and the average run-time to success, as well as the expected run-time of the algorithms in class **M1**. The data for the 500 and 1000-object problems is based only on the cleaned results, as described in Section 4.3.2. Thus we omit the MST/Random initialization and the 2000-object problem, since we lack enough reliable results in those cases.

The data shows that all configurations that use an intelligent initialization finish in less generations and less time than the corresponding algorithm with purely random initialization. But if the lower success rates are taken into account, when calculating the expected run-time, only few configurations turn out to be faster than random initialization. Even worse, no configuration shows reliably better results for all problem sizes. Thus we conclude that the introduction of intelligent initialization alone does not promise much success for improved scalability.

4.4.2 Using Improved Crossover and Hybridization

In Section 4.3.3 we have looked at algorithm configurations that reliably exhibit near-perfect success rates coupled with much faster run-times than the simple EA. To evaluate the scalability of those algorithms we will run further experiments on problems of even larger size for configurations that are most promising to achieve the best performance. The following subsections describe how we select those configurations, how we evaluate them, and present the observed results.

Selecting Configurations for further Evaluation

To get deeper insight on the scalability of the successful algorithm configurations it is required to evaluate them with larger-sized problems. In order to do this in a feasible time, we decide to select only a small subset of algorithm configurations.

In Section 4.3.3 the successful configurations have been categorized into three classes **S1** to **S3**. Out of those the algorithms in class **S2** showed a good fitness development for the best individual but a rather low average fitness in the population. This behavior has been attributed to the negative effects of applying the similarity heuristic hybridization to all individuals. Thus we will not further consider the algorithms in this class.

In the remaining classes we rank all configurations for each evaluated problem size based on the average run-time to success. We consider those configurations for further analysis that reach top performance for at least one problem size or show no statistically significant difference to the top performer. Table 4.10 gives the results of two-sample t-tests, which compare the fastest configurations to the top-performer for both classes **S1** and **S3**. We have not included the 100-object problems in this consideration; it is too easily solved and thus results in a very different ranking than the larger problem sizes, giving preference to simplicity over intelligent operators (cf. Appendix A).

We include algorithm configurations in the further evaluation if the t-test cannot reject the null hypothesis that the measured results originate from the same distribution as those of the top performer at a statistical significance of 1%. The p-Values that fall into this range are printed bold in Table 4.10. This results in the selection of six algorithm configurations for further evaluation:

- clust-xo-kb-hh, mut05-hill, heur-init
- clust-xo-kb-hh, mut05-hill, heur-rand-init
- clust-xo-kb-hh, mut05-hill, mst-init
- match-xo, mut05-hill, heur-init
- clust-xo-hh, mut05, heur-init
- clust-xo-hh, mut05, mst-init

Table 4.10: t-tests comparing configurations to the fastest algorithm in each group

Algorithms using the hill-climbing heuristic (class **S1**)				
Problem Size	Rank	Configuration	SRT	p
500	1	clust-xo-kb-hh, mut05-hill, heur-init	1.24	
	2	clust-xo-hh, mut05-hill, heur-init	1.27	$< 10^{-3}$
	3	match-xo, mut05-hill, heur-init	1.31	$< 10^{-21}$
	4	clust-xo-kb, mut05-hill, mst-init	1.32	$< 10^{-24}$
	5	clust-xo-kb-hh, mut05-hill, mst-init	1.34	$< 10^{-32}$
1000	1	clust-xo-kb-hh, mut05-hill, mst-init	1.83	
	2	match-xo, mut05-hill, heur-init	1.84	**0.521**
	3	clust-xo-kb-hh, mut05-hill, heur-init	1.84	**0.417**
	4	clust-xo-kb, mut05-hill, mst-init	1.89	0.002
	5	clust-xo-kb, mut05-hill, heur-init	1.89	$< 10^{-3}$
2000	1	clust-xo-kb-hh, mut05-hill, heur-rand-init	2.52	
	2	clust-xo-kb-hh, mut05-hill, heur-init	2.56	**0.010**
	3	clust-xo-kb-hh, mut05-hill, heur-mst-init	2.60	$< 10^{-6}$
	4	clust-xo-hh, mut05-hill, mst-init	2.62	$< 10^{-5}$
	5	clust-xo-hh, mut05-hill, heur-init	2.63	$< 10^{-8}$

Algorithms using Cluster-based Crossover (class **S3**)				
Problem Size	Rank	Configuration	SRT	p
500	1	clust-xo-hh, mut05, heur-init	1.31	
	2	clust-xo-kb, mut05, heur-rand-init	1.40	$< 10^{-25}$
	3	clust-xo-kb, mut05, mst-init	1.46	$< 10^{-43}$
1000	1	clust-xo-hh, mut05, heur-init	1.65	
	2	clust-xo-hh, mut05, heur-rand-init	1.76	$< 10^{-17}$
	3	clust-xo-hh, mut05, mst-init	1.77	$< 10^{-18}$
2000	1	clust-xo-hh, mut05, heur-init	2.95	
	2	clust-xo-hh, mut05, mst-init	2.96	**0.313**
	3	clust-xo-hh, mut05, heur-rand-init	3.02	$< 10^{-5}$

The updated Experimental Setup

We evaluate the selected algorithms with an experimental setup that closely resembles the system we have used earlier and described in Section 4.2. But in the meantime our hardware platform was updated to a 64 Bit Fedora 10 Linux and equipped with a total of 16 GB RAM.

We also choose to update the measurement code which previously recorded the elapsed real-time. Elapsed time is now measured based on the elapsed CPU-time of the process, as it is reported by a call to `getProcessCpuTime` of the `com.sun.management.OperatingSystemMXBean` interface [67]. This change aims to provide more robust measurements, independent from other system load that executes in parallel.

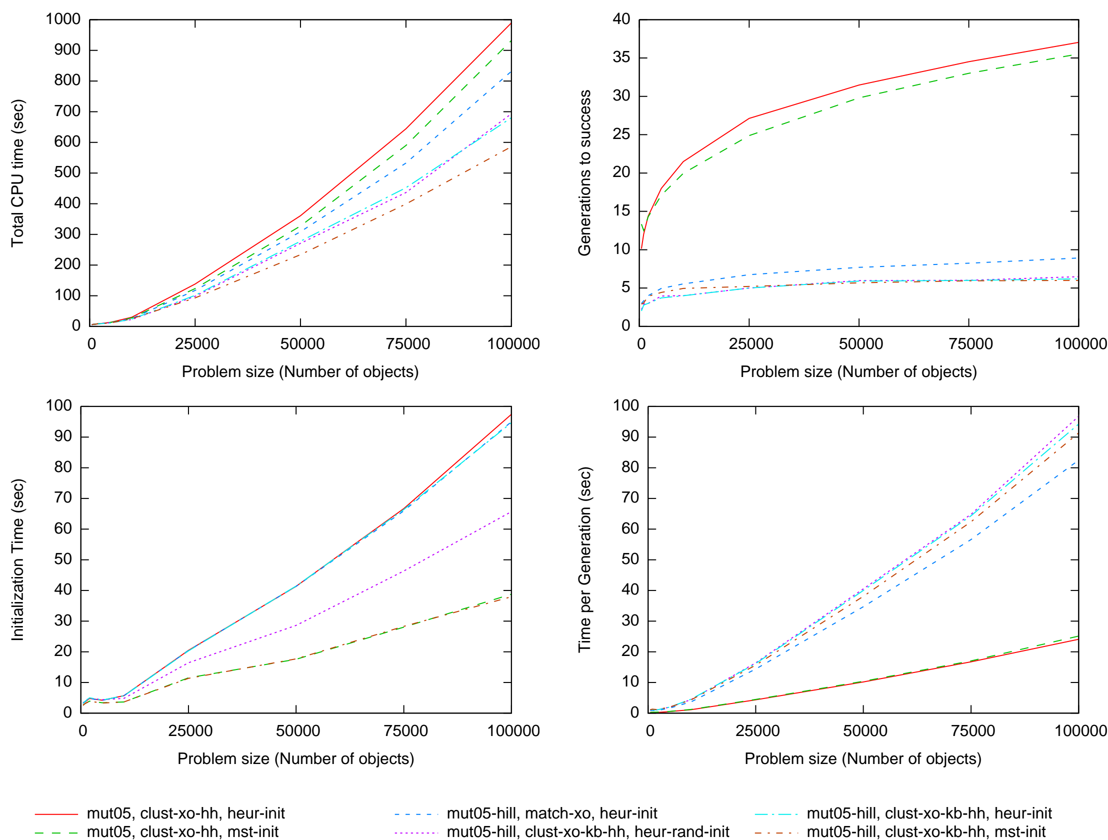


Figure 4.6: Run-times and required generations for the improved evolutionary algorithms

We conduct experiments again on 10-cluster problems, with problem sizes of 500, 1,000, 2,000, 5,000, 10,000, 25,000, 50,000, 75,000, and 100,000 objects. Even though we already measured data for the 3 smallest sizes, we evaluate the selected algorithms again to allow for comparability with the updated experimental setup.

Scalability of the improved Algorithms

We have evaluated the performance of the selected algorithm configurations using the experimental setup described above. Tables A.2 and A.3 on page 96 present the measured run-times for initialization and searching and the required generations to success. During all evaluation runs only one run of the clust-xo-hh, mut05, mst-init configuration did not reach a successful fitness level in the problem category of size 100,000. All other configurations achieved a 100% success rate.

We find that the average number of generations that are required to reach a successful fitness grows sub-linearly (Figure 4.6 top right). It is especially low when using the hill-climbing heuristic together with the cluster-based crossover, a combination that finds an acceptable solution in at most 7 generations even for the largest problem size.

Listing 4.1: Java source used to test linear scalability in `problemSize`

```
1  // Generate sorted array of Object IDs
2  ArrayList<Integer> permute = new ArrayList<Integer>(problemSize + 1);
3  for (int i=0; i<problemSize; i++)
4      permute.add(i);
5
6  // Now permute them
7  int n = permute.size();
8  while (n > 1) {
9      int idx = rand.nextInt(n);        // 0 <= idx < n
10     n--;
11     int tmp = permute.get(n);
12     permute.set(n, permute.get(idx));
13     permute.set(idx, tmp);
14 }
```

The same cannot be said about the average run-time required for initialization (Figure 4.6 bottom left) or the average run-time spent in each generation (Figure 4.6 bottom right). In both cases the observed times increase faster than linear with increasing problem size; this is especially evident when comparing the slope for problems smaller than 10,000 objects with the slope for larger problem sizes.

The observed scalability is to be expected for the MST initialization, since it sorts all edges in the induced similarity graph when using Kruskal's Algorithm and thus cannot be faster than $\mathcal{O}(|E| \log |E|)$. But for the heuristic initialization and the time it takes to process a single generation, we would expect linear scalability in the number of objects, since all involved loops iterate over either the number of clusters (which we kept constant), the number of objects, or the non-zero similarity values (which are linear in the number of objects in our test problems).

To trace the cause of the more-than-linear increase in run-time we have carefully reviewed our source code and the ECJ framework on which it is built for any hidden time consumption that might explain the observed behavior. We could not find any explanation in the code and have thus decided to run a simple subset of the whole application to see if the cause lies with the execution environment. The code in Listing 4.1 creates a permuted array of `problemSize` object indices; it is used, e.g., to create the order in which objects are processed by the similarity heuristic or the hill-climbing hybridization. The code is obviously linear since it contains only two loops that iterate `problemSize` and constant time operations otherwise.

Figure 4.7 presents the measured CPU times for executing 500 repetitions of the permutation code. The results are averages of 15 runs, which we have performed twice: once measuring process CPU time and once measuring thread CPU time. When using the `OperatingSystemMXBean` to capture the total CPU time of the Java process, the

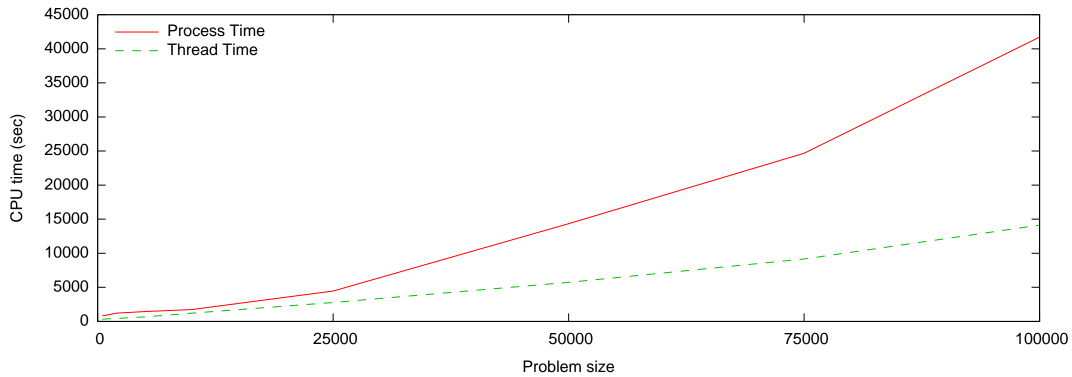


Figure 4.7: Measured CPU times of the complete JVM process and of the user thread only, for the $\mathcal{O}(n)$ benchmark in Listing 4.1

measurement not only includes time spent in user code but also all work that the Java Virtual Machine performs in helper threads, such as garbage collection or object finalization. When measuring CPU time with a `ThreadMXBean`, only the total time spent executing the main thread is considered. We see that the time spent in the main thread closely resembles a linear curve, while the total process time grows much faster.

This observation supports our belief that the non-linear scalability of the clustering EA, which we observed above, is not rooted in the algorithm itself but rather in the implementation and/or the execution environment. Further pinpointing the exact cause would exceed the scope of this work. A carefully crafted implementation in a lower-level language might very well scale linearly for the time spent in heuristic initialization and the search time used per generation. Of course, due to the increase in the required generations to success, the resulting algorithm will still scale faster than linear with respect to its total execution time. But the sub-linear increase in required generations is a vast improvement over the simple EA, whose required generations increase faster than linear.

5 Search Space Reduction by Multi-level Approaches

In the previous chapter we explored the incorporation of problem-specific knowledge into an evolutionary algorithm. We found several good algorithm configurations, which we ran successfully on problems of up to 100,000 objects. We now evaluate the possibility to further improve those algorithms by extending them to a multi-level system as described in Section 2.3.4.

Multi-level approaches have already been used for clustering problems. In Section 2.4.4, we presented two multi-level EAs for clustering. One of them *compresses* the search space by combining multiple objects to object-groups. The other *cuts* the search space into disjoint subsets. Both methods perform a first-level EA run on the problems of reduced size and then construct a final solution by running a second-level EA.

In this chapter, we propose two multi-level approaches, adapted from the reviewed literature. We design a system that *(i)* uses object-groups to compress a problem and one that *(ii)* cuts a problem into disjoint parts by snowball sampling. For each system, we describe the designed algorithm, perform initial experiments to find suitable parameters, and evaluate the configured methods on large problems.

For reasons of simplicity, we test the proposed approaches only with four successful algorithms from the previous chapter. We chose an unhybridized EA with cluster-based crossover (random selection, heuristic homeless assignment) and a hill-climbing hybridized EA with cluster-based crossover (keep-best selection, heuristic homeless assignment); both are combined with heuristic and MST-based initialization.

5.1 Using Object-Groups to Compress a Problem

Our first two-level evolutionary algorithm for clustering is inspired by a "search space reduction" technique of Gündüz-Öğüdücü and Uyar [37]. In their paper, they heuristically combine objects of a 4,000-object clustering problem to obtain a 1,000-object problem. First, their proposed EA is run for a predefined number of generations on the reduced problem. Thereafter, they restore the best solution of the first level to the full 4,000 objects and use it to seed 10% of the initial population for the second-level. The au-

thors reason that this allows to include high-quality individuals in the initial second-level population, while keeping a high diversity.

Our proposed approach follows a similar strategy. But we initialize the full second-level population with results of the first-level run; instead of using only the best first-level individual, we use the complete first-level population. This allows us to keep all information that was obtained at the first level and carries the diversity of the first-level population to the next level. Algorithm 5.1 presents our overall system, which contains two EA runs. We use the same EA configuration for both levels, except for the initial population. The system is compatible with all configurations that we presented in the previous chapter.

In the remainder of this section, we describe the details of the object-grouping heuristic, find appropriate parameters for the system, and compare the performance of the resulting two-level approach to the single-level EAs of the previous chapter.

5.1.1 The Object-grouping Heuristic

Algorithm 5.2 presents the search space compression, which combines objects of the original similarity matrix (SM) to groups of size *groupSize* that constitute the compressed similarity matrix SM'. Object-groups are created sequentially; i.e., a new group is only opened after the previous group has been filled. Each new group is seeded by an unassigned object (Line 7). The remaining objects are searched among the nearest neighbors of the current group members (Line 12); if all neighbors are already assigned to a group a random object is added (Line 14).

Once all objects have been assigned, the compressed similarity matrix, which stores similarity between pairs of object-groups, is built (Line 17). The similarity between two object groups $\mathfrak{o}_1 \subseteq V$ and $\mathfrak{o}_2 \subseteq V$ is set to

$$s(\mathfrak{o}_1, \mathfrak{o}_2) := \sum_{o_x \in \mathfrak{o}_1} \sum_{o_y \in \mathfrak{o}_2} s(o_x, o_y) \ . \tag{5.1}$$

The restoration of the search space (Algorithm 5.1, Line 4) is straightforward. We create one initial individual for the second-level EA out of each individual in the final first-level population. During the restoration, each object in the new initial individual is assigned the cluster number of its corresponding object-group.

5.1.2 Finding a Suitable Algorithm Configuration

Similar to the implementation by Gündüz-Öğüdücü and Uyar, we plan to run the first-level EA for a predefined number of generations. This leaves two open parameters for the

Algorithm 5.1: CompressingEA

```
1  begin
     Input: SimilarityMatrix SM, int groupSize, int gen
2    SM′ ← compress(similarityMatrix := SM, size := groupSize);
3    pop′ ← runEA(problem := SM′, generations := gen) ; /* First-level run */
4    pop ← uncompress(population := pop′);
5    runEA(problem := SM, initialPopulation := pop) ;   /* Second-level run */
6  end
```

Algorithm 5.2: compress

```
1  begin
     Input: SimilarityMatrix SM, int groupSize
     Output: SimilarityMatrix SM′
2    SM′.n ← ⌈SM.n / groupSize⌉;
3    SM.origSM ← SM;
4    SM′.objectMap[ ] ← new int[SM.n], initialized with -1;
5    Queue addOrder ← permute objects 0, ... n − 1 of SM;
6    for i = 0 to SM′.n − 1 do
       /* Seed the next object-group */
7      addObject ← dequeue unassigned object from addOrder;
8      SM′.objectMap[addObject] ← i;
9      PriorityQueue neighbors ← new PriorityQueue, initialize with the similarity
10         entries of addObject;
11     for j = 1 to groupSize − 1 do
         /* Add the nearest unassigned neighbors to the group */
12       addObject ← most similar unassigned neighbor in neighbors;
13       if addObject = null then
14         addObject ← dequeue unassigned object from addOrder;
15       SM′.objectMap[addObject] ← i;
16       neighbors.add(similarity entries of addObject);
     /* Build the similarity entries for SM′ */
17   foreach entry of SM.similarity do
18     i′ ← SM′.objectMap[entry.i];
19     j′ ← SM′.objectMap[entyr.j];
20     SM′.similarity(i′, j′)+ = entry.value;
21 end
```

two-level clustering EA: the number of generations in the first level and the object-group size. We design initial experiments to find a good configuration for both parameters.

We assume that the first level, which processes the compressed problem, is able to find medium-quality solutions to the original problem faster than a single-level EA, since the smaller genome should reduce runtime. We further assume that the first-level EA cannot find solutions of satisfactory quality to the original problem since it lacks the flexibility to assign objects in the same object-group to different clusters; this is likely to prevent the fine-tuning of a coarse solution.

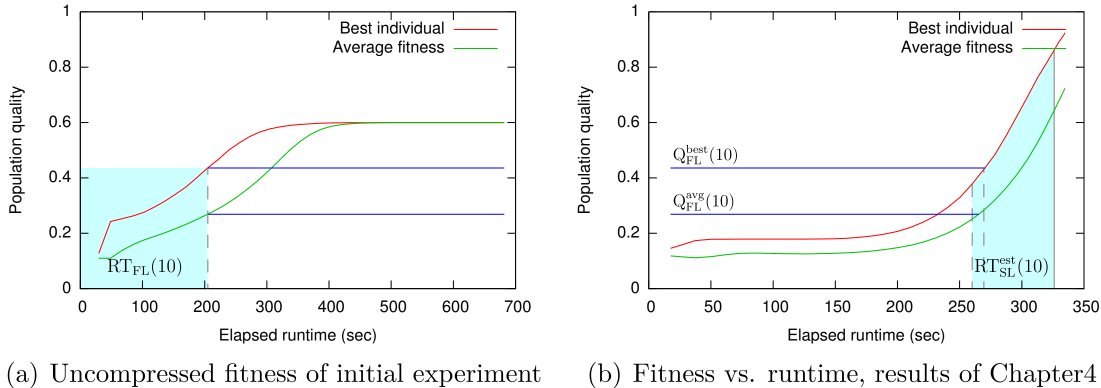
We need to determine a good cutoff point for the first-level run. If the first level is run too short, it might not yield the full benefit of finding a medium-quality fitness faster; if it is run too long, it might waste time on a medium-quality fitness that cannot be further improved. It is computationally infeasible to cover a large range of cutoff points through initial experiments with the two-level EA. Large-size test data sets of 200 problems take multiple hours or even days for evaluation. We thus select a cutoff point based on an estimated runtime, which we derive from a simplified experiment: We run the first level (FL) for a large enough number of generations to cover the interesting range of cutoff points. At each generation we observe the fitness of the uncompressed population. Based on this observation and the statistics for a single-level algorithm (EA), which we collected in Chapter 4, we estimate the runtime of the multi-level algorithm.

Simply put, our estimation assumes that the second-level run (SL) takes as long to succeed, as the single-level algorithm takes from the point of comparable quality. We set this point to the highest generation in the single-level run where average and best fitness are still below the respective values of the second-level initial population. Our calculation of the estimated runtime, $\mathrm{RT}^{\mathrm{est}}$, can be expressed as:

$$\begin{aligned} \mathrm{RT}^{\mathrm{est}}(g) &= \mathrm{RT}_{\mathrm{FL}}(g) + \mathrm{RT}^{\mathrm{est}}_{\mathrm{SL}}(g) && (5.2)\\ &= \mathrm{RT}_{\mathrm{FL}}(g) + \Big[\ \mathrm{RTQ}^{\geq}_{\mathrm{EA}}(0.85, \infty) - \mathrm{RTQ}^{<}_{\mathrm{EA}}\left(\mathrm{Q}^{\mathrm{best}}_{\mathrm{FL}}(g), \mathrm{Q}^{\mathrm{avg}}_{\mathrm{FL}}(g)\right)\ \Big]\ . && (5.3) \end{aligned}$$

$\mathrm{RT}_{\mathrm{FL}}(g)$ denotes the runtime the first-level spent up to generation g, $\mathrm{Q}^{*}_{\mathrm{FL}}(g)$ denotes the best/average fitness, when uncompressing the first-level population at generation g. $\mathrm{RTQ}^{\geq}_{\mathrm{EA}}(best, avg)$ denotes the runtime spent by the single-level algorithm up to the first time that the average or best fitness exceed or equal the argument values, $\mathrm{RTQ}^{<}_{\mathrm{EA}}(best, avg)$ denotes the runtime spent by the single-level algorithm up to the last time that the average and best fitness fall below the argument values.

Figure 5.1 gives an example for the $\mathrm{RT}^{\mathrm{est}}$ calculation. Figure 5.1(a) shows the fitness of the uncompressed population vs. elapsed runtime that we found in the experiment. The horizontal lines mark the average and best fitness after generation 10. Figure 5.1(b) is an averaged fitness plot of a single-level EA with the same configuration, based on the experiments we ran in Chapter 4. In generation 23, the plotted fitness of the best individual



(a) Uncompressed fitness of initial experiment (b) Fitness vs. runtime, results of Chapter4

Figure 5.1: Exemplary calculation of the estimated runtime for the two-level EA based on object-groups. The fitness of the first-level run in generation 10 is marked. The corresponding fitness in the single-level EA falls between generations 22 and 23 (dashed lines)

falls below the corresponding marker, but the average fitness is still higher. Generation 22 is the highest generation for which both plotted values are lower than the marked levels. Thus the average runtime of generation 22 defines $RTQ^{<}_{EA}\left(Q^{best}_{FL}(10), Q^{avg}_{FL}(10)\right)$ and the average runtime of generation 23 defines $RTQ^{\geq}EA\left(Q^{best}_{FL}(10), Q^{avg}_{FL}(10)\right)$.

To conduct the initial tests, we uncompress the population after each generation and report statistics on the uncompressed population. The computational time used for the uncompressing and reporting is not attributed to the runtime of the first-level EA. We test four successful configurations of Chapter 4 with varying group sizes. The tested group sizes are 5, 10, 50, and 100 objects per group. Based on the average generations a single-level EA requires to succeed (Figure 4.6), we limit the interesting range of cutoff points to 25 generations for unhybridized configurations and to 7 generations for hybrid EAs. Due to time constraints, those settings are tested on a limited selection of problem sizes: 10,000, 25,000, 50,000, and 75,000 objects.

Table 5.1 shows the estimated runtimes that result from the initial experiments. For each algorithm configuration and each combination of object-group size and problem size, the fastest estimated runtime is presented alongside the cutoff generation for which it was achieved. We find that independent of the specific algorithm configuration and the problem size, an object-group size of 100 objects always results in the fastest estimated runtime. We decide to fix this group size for the final evaluation.

As for the best cutoff generation, we see similarities between algorithm configurations with the same initialization method. The majority of heuristically initialized runs achieves the best estimated runtime for a first-level EA that runs two generations. The group of MST initialized configurations shows a strong tendency towards fewer generations and achieves most often the fastest runtime for 0 first-level generations. This

Table 5.1: Results of initial experiments to determine the fastest configuration for the object-grouping two-level EA. The tables show the fastest estimated runtimes (in seconds) and their associated cutoff generation for all combinations of group size and problem size.

(a) clust-xo-hh, mut05, heur-init

	10,000 objects		25,000 objects		50,000 objects		75,000 objects	
Group Size	Cutoff	RT^{est}	Cutoff	RT^{est}	Cutoff	RT^{est}	Cutoff	RT^{est}
5	4	29.6	2	126.9	2	340.3	0	620.0
10	5	31.6	5	128.5	7	338.2	8	605.4
50	0	26.2	2	96.3	4	239.4	3	417.9
100	0	21.1	1	87.7	2	223.3	2	387.2

(b) clust-xo-hh, mut05, mst-init

	10,000 objects		25,000 objects		50,000 objects		75,000 objects	
Group Size	Cutoff	RT^{est}	Cutoff	RT^{est}	Cutoff	RT^{est}	Cutoff	RT^{est}
5	3	27.6	3	95.6	3	214.6	3	354.5
10	4	27.6	4	92.3	3	200.2	3	352.8
50	0	16.4	0	49.5	0	184.7	3	336.0
100	0	15.0	0	44.2	0	111.3	0	221.9

(c) clust-xo-kb-hh, mut05-hill, heur-init

	10,000 objects		25,000 objects		50,000 objects		75,000 objects	
Group Size	Cutoff	RT^{est}	Cutoff	RT^{est}	Cutoff	RT^{est}	Cutoff	RT^{est}
5	2	21.1	3	80.9	3	210.7	3	277.5
10	2	20.8	3	74.2	2	171.1	3	261.5
50	0	23.0	2	65.9	2	150.0	2	248.6
100	0	19.9	2	62.1	2	137.7	2	230.7

(d) clust-xo-kb-hh, mut05-hill, mst-init

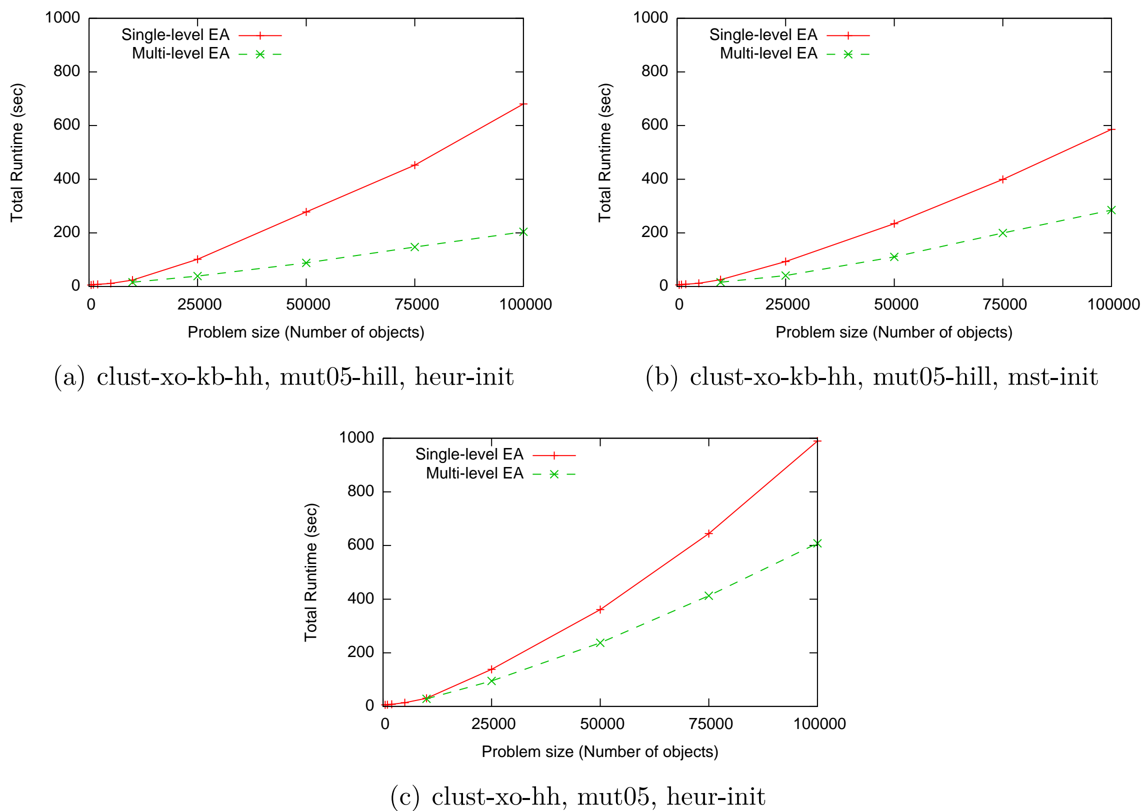
	10,000 objects		25,000 objects		50,000 objects		75,000 objects	
Group Size	Cutoff	RT^{est}	Cutoff	RT^{est}	Cutoff	RT^{est}	Cutoff	RT^{est}
5	2	25.1	3	75.4	3	191.6	1	349.3
10	1	22.2	2	59.5	2	153.5	2	264.1
50	0	17.9	0	48.0	2	140.1	2	241.0
100	0	15.3	0	46.6	2	130.4	2	231.5

corresponds to initializing a single-level EA by an MST of the compressed similarity matrix. This indicates that the first-level EA cannot achieve much significant improvement that is worth to spend time on. Only on very large problems does a first-level run of 2 generations result in a better estimated runtime.

We decide to fix the first-level generations at 2 for heuristically initialized configurations. For MST initialized ones, we run the first-level EA for 1 generation, as a compromise between 0 and 2 being the best cutoff points.

5.1.3 Performance of the Approach

We have evaluated the object-grouping multi-level system in combination with four different EA configurations on a range of large-scale problems. The complete results are given in Table B.1 on page 99. Three of the four configurations show a 100% success rate; their runtimes (averaged over 200 test problems) are visualized in Figure 5.2 and compared to the runtimes of a single-level EA with the same configuration.



(a) clust-xo-kb-hh, mut05-hill, heur-init

(b) clust-xo-kb-hh, mut05-hill, mst-init

(c) clust-xo-hh, mut05, heur-init

Figure 5.2: Comparison of the object-grouping multi-level EAs to the single-level EAs of the previous chapter.

We find that for unhybridized EAs, the presented runtime estimates underestimated the time required for the second level. For example, the heuristically initialized configuration had an estimated runtime of 387 seconds for 75,000 objects, but the true SRT is above 400 seconds. Still, the true runtimes are an improvement over the corresponding single-level EA. The multi-level system is able to reduce the required runtime by up to 40%.

The unhybridized, MST initialized EA shows a worse result: the multi-level version frequently converges to local optima, which results in low success rates in the range of 6% to 20%. SRT decreases by at most 20% compared to the single-level EA. This renders the clust-xo-hh, mut05, mst-init object-grouping EA infeasible for successful clustering.

For the hill-climbing hybridized EAs, we find that the true runtimes have been overestimated. All second-level runs find a solution of acceptable quality in at most 2 generations. Both initialization techniques result in successful EAs; the heuristically initialized configuration achieves a consistently better SRT than the MST initialized one. Compared to a single-level EA, the heuristically initialized EA is able to reduce the required runtime by up to 70%.

In short, we can say that the object-grouping multi-level system allows respectable performance improvements at the cost of reduced robustness. For example, in combination with the clust-xo-kb-hh, mut05-hill, heur-init configuration, the two-level approach sets a new record SRT for the 100,000-object problem of only 200 seconds; but in combination with the clust-xo-hh, mut05, mst-init configuration, the two-level approach is unsuccessful and frequently trapped in local optima.

5.2 Cutting a Problem by Snowball Sampling

Our second two-level evolutionary algorithm for clustering is inspired by a “search space division” technique of Korkmaz [54]. In his paper, he randomly cuts a feature-based data set into s subsets, in order to handle a 700-object data set with an EA that had before only been used for problems of up to 150 objects. Each subset is clustered by an independent multi-objective EA run for a predefined number of generations. Thereafter, he determines a suitable number of clusters, c, based on characteristics of the Pareto fronts. The author notes that any number equal to or larger than the true number of clusters is suitable for c. In the second level, he processes the $c \cdot s$ partial clusters of all subproblems by the clustering MOEA to determine which ones should be merged.

We design a similar two-level approach, but change certain details to better suit our test problems and previous algorithms. Our test data is represented as a sparse similarity matrix, thus not all object pairs are connected in the induced similarity graph. If we cut

Algorithm 5.3: CuttingEA

```
1 begin
    Input: SimilarityMatrix SM, int numParts, int gen
2   SM'[ ] ← cut(similarityMatrix := SM, parts := numParts);
3   for i = 0 to numParts − 1 do                    /* First-level runs */
4       pop'[i] ← runEA(problem := SM'[i], generations := gen);
5   pop ← splice(population := pop'[ ]);
6   runEA(problem := SM, initialPopulation := pop) ;   /* Second-level run */
7 end
```

the set of objects randomly, most likely each subset would be highly disconnected. To counter this, we use snowball sampling to create the subsets.

The second-level EA in the original paper is restricted to process only the partial clusters. Thus objects that have been grouped together in the first level cannot be separated. We follow a similar strategy as in the previous section to increase flexibility of the second level. The results of the clustered subsets are combined and seed the initial population for a fully flexible second-level EA. Algorithm 5.3 summarizes our overall approach.

In the remainder of this section we describe the details of snowball sampling used to cut the data and propose two splicing heuristics to recombine partial clusters. We find appropriate parameters for the system, evaluate its performance and compare it to the single-level EAs of the previous chapter.

5.2.1 The Snowball Sampling

Algorithm 5.4, which is based on snowball sampling, divides a given problem into multiple parts. Snowball sampling originates from social sciences. It refers to a sampling scheme in which a set of initial candidates is instructed to recruit their friends, which in turn recruit their friends and so on. In an information technology environment, this sampling process has been used, e.g., in social network studies [62].

We use a snowball sampling approach instead of a random division of the problem, because of the sparse test problems. The test problems have an expected node degree of 10 in the induced similarity graphs (cf. Section 3.2). Thus, a random division of the search space bears the high risk to detach objects from all their neighbors. Our version of snowball sampling is designed to reduce this risk.

Algorithm 5.4 seeds each subset with a random object (Line 5) and progresses the subset construction from the selected seeds. In its original context, snowball sampling is usually performed with multiple initial candidates, who are all asked to refer multiple friends. We simplify this scheme so that each object refers only one neighbor, the most

Algorithm 5.4: cut

```
1  begin
       Input: SimilarityMatrix SM, int parts
       Output: SimilarityMatrix SM′[ ]
2      partID[ ] ← new int[SM.n], initialized with -1;
3      Queue addOrder ← permute objects 0, ... n − 1 of SM;
4      Stack backtrack[ ] ← new Stack[parts];
5      for i = 0 to parts − 1 do                      /* Seed each snowball */
6          addObject ← dequeue unassigned object from addOrder;
7          partId[addObject] ← i;
8          backtrack[i].push(addObject);
9      while not all objects are assigned do   /* Grow the snowballs in parallel
       */
10         for i = 0 to parts − 1 do
               /* Search a candidate among neighbors of snowball objects  */
11             while backtrack[i] not empty do
12                 addObject ← most similar unassigned neighbor of top element;
13                 if addObject = null then backtrack[i].pop() else break;
14             if addObject = null then
15                 addObject ← dequeue unassigned object from addOrder;
16             partId[addObject] ← i;
17             backtrack[i].push(addObject);
       /* Copy the similarity entries to the SM′[ ] members                */
18     foreach entry of SM.similarity do
19         if partId[entry.i] = partId[entry.j] then
20             SM′[partId[entry.i]].similarity(entry.i, entry.j) ← entry.value;
21 end
```

similar one (Line 12). The referred object is added to the same subset and refers the next object, and so on. Only if an object has no more unassigned neighbors, we backtrack until a feasible connection is found (Line 13).

We hope this approach results in a cut for which holds: *(i)* subsets are connected, *(ii)* the neighborhood of an object contains nodes that belong to the same cluster, and *(iii)* each subset contains a good mix of objects from all clusters. *(i)* is supported by the basic behavior of the sampling process: new objects are found along edges in the similarity graph; an unconnected object is only added as the last resort (Line 14). *(ii)* is achieved since we give preference to high similarity edges, which are likely to refer an object of

the same cluster as the current referee. *(iii)* is a desirable goal, so that the first-level EA can find a clustering that fits the overall problem. The goal seems contradicting to *(ii)*; but the available edge with the highest quality does not necessarily lead to an object of the same cluster (otherwise the clustering task would be trivial). Thus, the snowball creation is able to "switch" to an object of a different cluster; once this occurs, the newly involved cluster is likely to be further sampled for a while, until another switch occurs.

5.2.2 The Splicing Heuristics

The first-level EA processes each subset of the problem independently. Thereafter, we construct an initial population for the second-level EA that is based on the clusters of the subsets. We propose to select one individual per subset population and splice them in order to form a single new solution candidate for the second level. We select the individuals from the subset populations according to their fitness, starting with the best. This assures that the first second-level solution candidate is combined of the best solutions in each subset, the second one is combined of all second-best solutions, and so on.

For s subsets and a k-clustering problem, the splicing is presented with $s \cdot k$ partial clusters, which need to be combined to a single k-cluster solution. We propose two heuristics for this combination, which are based on the summed similarity between each pair of partial clusters. The *unrestricted splicing heuristic* is free to combine the partial clusters in any possible way to form k clusters. The *restricted splicing heuristic* forms each of the k clusters out of exactly one partial cluster per subset.

The Unrestricted Splicing Heuristic

The unrestricted approach uses single-linkage clustering as a heuristic to combine the partial clusters. Pairs of partial clusters are processed in descending order of their pairwise similarity; for each pair of partial clusters, the components in which they are contained are merged, until only k clusters are left. This corresponds to building an MST-clustering on the fully connected graph of partial clusters.

The Restricted Splicing Heuristic

The restricted approach is a modified single-linkage splicing, which is constrained to only combine partial clusters that belong to different parts. The restricted splicing algorithm utilizes *components*, a data structure that describes how to build a final cluster out of partial clusters. Similar to the unrestricted approach, pairs of partial clusters are processed in descending order of their pairwise similarity; but the components in which

they are contained are only merged if they are compatible, meaning that the resulting combination would not contain two partial clusters that belong to the same part.

For example, for $k = 3$ and $s = 3$, the partial cluster 2 in part number 1 is initialized with the component (-2_1-), partial cluster 0 in part 2 is initialized with $(--0_2)$. If the pair $2_1, 0_2$ has the highest pairwise similarity, their components are combined to form $(-2_1 0_2)$. The second similar pair might be $1_0, 1_2$ and the algorithm forms the component $(1_0 - 1_2)$. If the third similar pair is $1_0, 2_1$, the corresponding components are incompatible, since they both contain a partial cluster for part 2. The algorithm cannot combine them and advances to the next pair of partial clusters.

The restricted splicing heuristic processes all pairs of partial clusters, building larger and larger components. Once all positions in a component are filled, it is closed and forms an initial cluster for the second-level solution candidate. Above procedure does not guarantee that all components are closed, once all pairs of partial clusters have been processed. For example, the exemplary situation above might result in the final components $(2_0 1_1 2_2)$, $(1_0 - 1_2)$, $(-2_1 0_2)$, and $(0_0 0_1 -)$. Since $k = 3$, there is one excessive component, but no further combinations are possible. In this case, the heuristic splits up excessive components and redistributes their partial clusters among the remaining open components on a first fit basis. This allows to close all remaining components and a valid solution candidate for the second level has been built.

5.2.3 Finding a Suitable Algorithm Configuration

The proposed two-level approach has open parameters for which we need to find a suitable configuration. As for the object-grouping EA, the number of generations for the first level is a variable. Further parameters are the number of subparts in which we cut the problem and the decision between the two proposed splicing heuristics.

As before, we perform initial experiments to determine the parameters. This time we are more skeptical about the suitability of the designed approach: Even though the snowball sampling is intended to cut a problem into “meaningful” subparts, there is the risk that too much similarity information is lost to allow for a good clustering. Thus, we run more conservative initial experiments: The first-level EA is run for a number of generations sufficient to achieve fitness convergence in the subproblems. We then test whether the resulting initial configuration leads to an improved second-level run.

We evaluate configurations with a variable number of subparts in combination with both splicing heuristics. Due to time constraints, we limit ourselves to the 50,000-object problem in the initial experiments. The initial experiments are conducted for 2, 5, and 10 subparts. The first level is run for 30 generations in combination with the unhybridized EA and for 7 generations in combination with the hill-climbing hybridization.

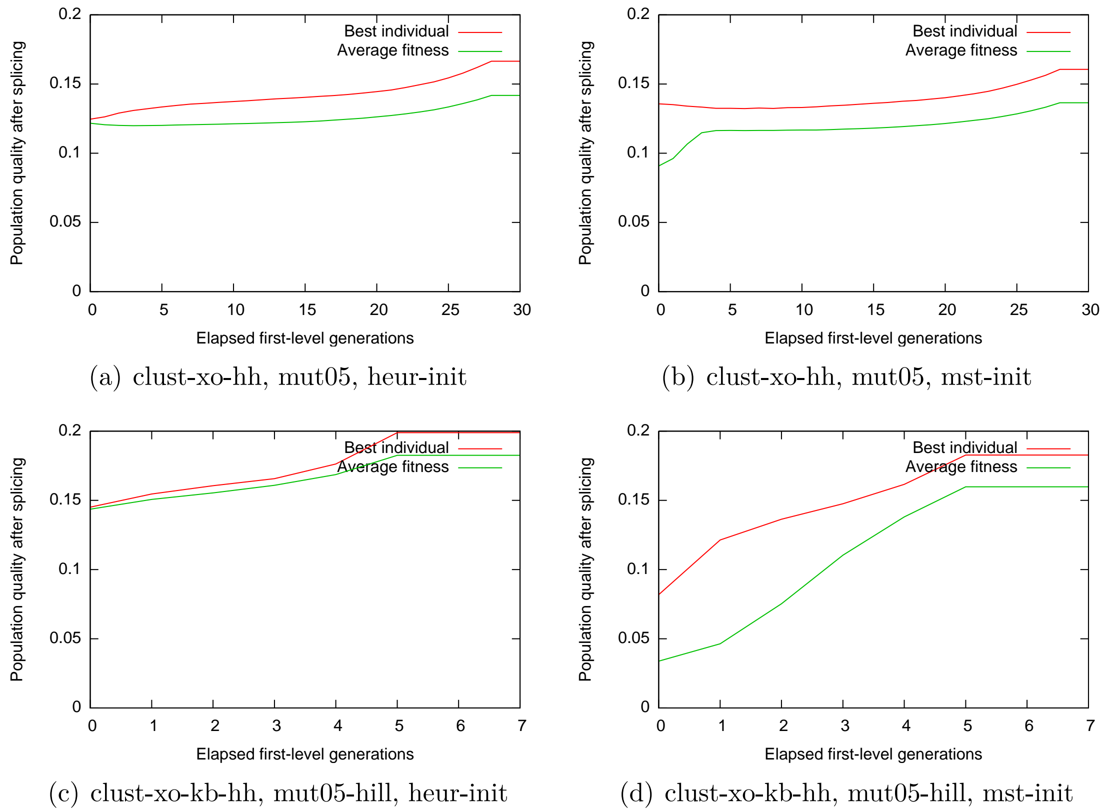
Table 5.2: Average generations to success spent in the second level of cutting EA for 50,000-object problem. The total generations of a single-level EA are shown for comparison.

		clust-xo-hh, mut05		clust-xo-kb-hh, mut05-hill	
Subparts	Splicing	heur-init	mst-init	heur-init	mst-init
2	unrestricted	17.2	18.8	2.2	8.9
2	restricted	14.8	15.5	2.0	2.5
5	unrestricted	32.8	32.7	5.0	8.3
5	restricted	30.8	30.8	4.8	4.0
10	unrestricted	32.6	32.8	5.1	5.9
10	restricted	34.1	34.4	5.2	4.9
Single-level EA		31.5	29.8	5.9	5.7

The combination of snowball sampling with MST initialization showed a problem during the experiments: if a snowball is re-seeded because its backtracking stack emptied (Algorithm 5.4, Line 14), the resulting subproblem is not connected. This leads to a failure of the MST initialization, since too many connected components are discovered. In a productive environment this could be fixed in two ways. An updated MST initialization could check the number of connected components in the input problem and remove less than $k - 1$ edges; or an updated cutting algorithm could insert an explicit edge with weight zero to connect multiple seeds in a single snowball. For the purpose of analyzing the suggested two-level approach, we decide to ignore the erroneous problems and report results only on unproblematic runs. In every MST initialized configuration at most 25% of all runs are affected.

Table 5.2 presents the average generations that the two-level EA has spent in the second level. We compare those to the time a single-level EA takes for the same problem class to infer a configuration for the final evaluation. With respect to the splicing heuristics, the restricted splicing is superior to the unrestricted splicing in most cases. For restricted splicing, we see a consistent increase in the required generations with an increasing number of subparts. Thus we choose a 2-subpart configuration and restricted splicing for further evaluation. Other configurations are not very promising and some even worsen the average generations required for the second-level run to succeed.

The final open parameter is the number of generations for the first-level run. The selected configuration improves the average generations to success by 14.3 or 16.7 for unhybridized and by 3.2 or 3.9 for hybridized algorithms. Thus the effort of spending 30 or 7 generations on the first level respectively, as was done for the initial experiment, is too high to achieve an overall decrease of runtime. Figure 5.3 presents the population fitness that was recorded in the initial experiment, similar to the object-grouping approach:



(a) clust-xo-hh, mut05, heur-init

(b) clust-xo-hh, mut05, mst-init

(c) clust-xo-kb-hh, mut05-hill, heur-init

(d) clust-xo-kb-hh, mut05-hill, mst-init

Figure 5.3: Fitness development of the resulting population, when applying the restricted splicing heuristic to a 2-subpopulation first-level run

after each generation the splicing heuristic was applied and the resulting population was evaluated. We see a very regular increase in fitness, and no obvious candidate for the number of first-level generations can be inferred from the plots. In order to start the second level with a good initial population, we decide to set the number of first-level generations rather high but still below the gain we have over a single-level EA. We choose to run the first-level EA for 10 generations for unhybridized configurations and 3 generations for hybridized ones.

5.2.4 Performance of the Approach

We have evaluated the cutting multi-level system in combination with four different EA configurations on a range of large-scale problems. The complete results are given in Table B.2 on page 100. All configurations reach success rates above 99%. But in conjunction with the selected parameters for the cutting system, no EA configuration is able to outperform a single-level approach with respect to the average runtime.

In all configurations, the average number of generations that the second level requires until success, is lower than the generations required by a corresponding single-level EA. But the attainable reduction is not large enough to make up for the computational time spent on the first level. For example, for the 50,000-object problems we see a benefit of 1.6 or 3.8 generations for unhybridized configurations and 1.6 or 1.9 generations for hybridized configurations. These values are below the ones of the initial experiments, due to the reduced number of first-level generations.

There is the possibility that a different number of first-level generations allows the cutting multi-level EA to achieve a faster SRT than a corresponding single-level EA. But in the light of the experimental results we have seen so far, we expect no noteworthy improvement.

The worse performance in comparison to the object-grouping approach is reasonable: In the object-grouping approach, a single first-level run is performed on a problem of size $\frac{n}{\text{group-size}}$; due to the reduced genome size, the first-level EA is able to process a certain amount of generations faster than an EA that works on the uncompressed problem. The cutting approach, on the other hand, performs s runs on problems of size $\frac{n}{s}$ for a problem that is cut into s subproblems. Thus, the effort to process a certain amount of generations once per subproblem is approximately the same as processing the same number of generations in a single-level EA that works on the complete problem. Speedup can only result from a faster convergence in the first-level runs, which might result from the reduced problem size. This was the case in the original paper [54] for feature-based data. But for our sparse pairwise similarity problems, the subproblems do not carry enough information in order to quickly converge to a good clustering.

6 Summary, Conclusion and Outlook

In this thesis, we have discussed scalability issues of evolutionary algorithms. We presented a general survey on this topic and found clustering to be an interesting problem for our own studies. Based on our understanding of scalability as the behavior of performance over changing problem sizes, we designed scalable test data for the pairwise similarity-based clustering problem. A simple EA, which we introduced as the base algorithm, exhibited an unfavorable scalability; runtimes increase more than quadratically with growing problem size. With an average runtime to success of more than 20 minutes for a 2,000-object problem, the base algorithm is unsuitable for large problems.

We proposed several modifications to the base algorithm in the form of problem-specific knowledge in custom initialization and crossover operators and hybridizations with clustering heuristics. We evaluated various combinations of the suggested modifications: In total, 126 different clustering EAs have been tested on problems of up to 2,000 objects. We found that intelligent initialization alone, used in an unhybridized algorithm that employs traditional crossover, cannot improve scalability. However, algorithms with a cluster-based crossover or a hill-climbing hybridization showed pleasing performance. For example, the 2,000-object problem could be solved in an average time of less than three seconds. Furthermore, runtimes were found to increase in a near-linear manner with growing problem size. We could successfully solve problems of up to 100,000 objects with an average runtime below 1,000 seconds.

Based on two-level EAs in the literature, we extended the successful EA configurations to follow a similar approach. Our proposed methods first process a size-reduced problem with a clustering EA; they then use the resulting population to seed a second-level EA. We suggested two approaches for the size reduction: search space compression by summarization and cutting the input problem into multiple smaller subproblems for the first level. Experimental evaluation showed that the summarization by grouping multiple objects has the potential for further runtime reduction. The alternative approach to cut the search space into subproblems had no positive effect. We tested the object-grouping system in combination with good algorithm configurations that we found for the single-level EA. We learned that the multi-level approach decreases robustness, since some of the configurations consistently converged to local optima only, but also allows substantial runtime improvements. The best multi-level configuration consistently discovered solutions of acceptable quality in only 30% of the time required by the corresponding

single-level EA; the 100,000-object problems were solved with an average runtime of 200 seconds.

We can conclude that, in the studied problem domain, evolutionary approaches can successfully serve as the basis for well-scaling methods. But this requires an appropriate integration of problem-specific knowledge. Although a standard EA is flexible enough to be quickly adapted for clustering, standard operators alone are not sufficient to achieve good performance or scalability and make it infeasible to solve large problems.

If large problem sizes need to be handled with little effort, the development of problem-specific evolutionary operators might be too costly. In such situations, we can advise to base algorithm design on a problem-specific search heuristic, even if it is prone to fail by getting stuck in local optima, and then hybridize the heuristic with a simple EA. We had a good experience with such hybrid EA configurations: they outperformed a standard EA in terms of runtime and surpassed the success rates of non-evolutionary heuristic use.

If performance is a main objective, the hybrid EA can be further optimized by problem-specific operators or multi-level approaches. These optimizations allow substantial performance improvements but require more effort during their development and increase the risk of failure due to a decreased robustness of the resulting algorithm.

In this work, we could only cover the experimental evaluation on a set of synthetically generated test problems. In order to support our results and to broaden the insight on clustering EAs, further experiments on real-world data sets will be required. Not in the scope of this work were general techniques for improving EA performance and their applicability to the clustering problem. The usage of parallelization is certainly possible and, in the master/slave model, will allow a predictable speedup. Attention should be given to the prospects of applying linkage learning techniques to large clustering problems. Because of the high genome interdependency, clustering poses an extremely tough problem for linkage learning and studies in this direction might yield interesting results.

A Experimental Results for Chapter 4

This appendix presents the detailed results of the experiments in Chapter 4.

A.1 The Initial Evaluation of All Configurations

We have evaluated various algorithm configurations in Chapter 4, which we grouped into a total of 13 classes (Table 4.4 on page 54). This appendix presents the runtimes and success rates for all classes in Table A.1 on the next page.

Each algorithm was evaluated on four problem sizes, ranging from 100 objects to 2000 objects; 200 independent algorithm runs have been performed in each category. For each size, we give the achieved success rate (SR), the average run-time to success (SRT), and the expected run-time (ERT), as defined in Section 3.3.1.

Table A.1: Success Rate (SR), average run-time for successful runs (SRT) and expected run-time (ERT) for all algorithm configurations. Each configuration was run on 200 problems for each problem size. We have evaluated 100, 500, 1000, and 2000 object-problems. The case where SR is zero is left as blanks.

				100			**500**			**1000**			**2000**		
Class	Mutation	Crossover	Initialization	**SR**	**SRT**	**ERT**	**SR**	**SRT**	**ERT**	**SR**	**SRT**	**ERT**	**SR**	**SRT**	**ERT**
S1	mut05-hill	clust-xo	heur-init	1.00	0.54	0.54	1.00	1.35	1.35	1.00	2.00	2.00	1.00	3.03	3.03
	mut05-hill	clust-xo	heur-rand-init	1.00	0.56	0.56	1.00	1.43	1.43	1.00	2.09	2.09	1.00	3.13	3.13
	mut05-hill	clust-xo	heur-mst-init	1.00	0.54	0.54	1.00	1.51	1.51	1.00	2.15	2.15	1.00	3.27	3.27
	mut05-hill	clust-xo	heur-mst-rand-init	1.00	0.56	0.56	1.00	1.57	1.57	1.00	2.27	2.27	1.00	3.33	3.33
	mut05-hill	clust-xo	rand-init	1.00	0.52	0.52	1.00	1.39	1.39	1.00	2.09	2.09	1.00	3.32	3.32
	mut05-hill	clust-xo	mst-init	1.00	0.47	0.47	1.00	1.37	1.37	1.00	2.05	2.05	1.00	3.11	3.11
	mut05-hill	clust-xo	mst-rand-init	1.00	0.53	0.53	1.00	1.51	1.51	1.00	2.16	2.16	1.00	3.23	3.23
	mut05-hill	clust-xo-hh	heur-init	1.00	0.53	0.53	1.00	1.27	1.27	1.00	1.89	1.89	1.00	2.63	2.63
	mut05-hill	clust-xo-hh	heur-rand-init	1.00	0.54	0.54	1.00	1.39	1.39	1.00	1.98	1.98	1.00	2.78	2.78
	mut05-hill	clust-xo-hh	heur-mst-init	1.00	0.52	0.52	1.00	1.47	1.47	1.00	2.04	2.04	1.00	2.85	2.85
	mut05-hill	clust-xo-hh	heur-mst-rand-init	1.00	0.55	0.55	1.00	1.53	1.53	1.00	2.13	2.13	1.00	2.97	2.97
	mut05-hill	clust-xo-hh	rand-init	1.00	0.54	0.54	1.00	1.41	1.41	1.00	2.11	2.11	1.00	2.89	2.89
	mut05-hill	clust-xo-hh	mst-init	1.00	0.48	0.48	1.00	1.41	1.41	1.00	1.98	1.98	1.00	2.61	2.61
	mut05-hill	clust-xo-hh	mst-rand-init	1.00	0.53	0.53	1.00	1.52	1.52	1.00	2.07	2.07	1.00	2.82	2.82
	mut05-hill	clust-xo-kb	heur-init	1.00	0.55	0.55	1.00	1.34	1.34	1.00	1.89	1.89	1.00	2.85	2.85
	mut05-hill	clust-xo-kb	heur-rand-init	1.00	0.55	0.55	1.00	1.42	1.42	1.00	1.98	1.98	1.00	2.86	2.86
	mut05-hill	clust-xo-kb	heur-mst-init	1.00	0.54	0.54	1.00	1.49	1.49	1.00	2.03	2.03	1.00	2.92	2.92
	mut05-hill	clust-xo-kb	heur-mst-rand-init	1.00	0.56	0.56	1.00	1.53	1.53	1.00	2.17	2.17	1.00	3.00	3.00
	mut05-hill	clust-xo-kb	rand-init	1.00	0.52	0.52	1.00	1.35	1.35	1.00	1.97	1.97	1.00	3.08	3.08
	mut05-hill	clust-xo-kb	mst-init	1.00	0.48	0.48	1.00	1.32	1.32	1.00	1.89	1.89	1.00	2.77	2.77
	mut05-hill	clust-xo-kb	mst-rand-init	1.00	0.52	0.52	1.00	1.43	1.43	1.00	2.07	2.07	1.00	2.97	2.97
	mut05-hill	clust-xo-kb-hh	heur-init	0.99	0.59	0.62	1.00	1.24	1.24	1.00	1.84	1.84	1.00	2.56	2.56
	mut05-hill	clust-xo-kb-hh	heur-rand-init	1.00	0.55	0.55	1.00	1.34	1.34	1.00	2.00	2.00	1.00	2.52	2.52
	mut05-hill	clust-xo-kb-hh	heur-mst-init	1.00	0.54	0.54	1.00	1.43	1.43	1.00	1.98	1.98	1.00	2.60	2.60
	mut05-hill	clust-xo-kb-hh	heur-mst-rand-init	1.00	0.56	0.56	1.00	1.50	1.50	1.00	2.11	2.11	1.00	2.67	2.67
	mut05-hill	clust-xo-kb-hh	rand-init	1.00	0.55	0.55	1.00	1.42	1.42	1.00	1.97	1.97	1.00	2.85	2.85
	mut05-hill	clust-xo-kb-hh	mst-init	1.00	0.48	0.48	1.00	1.34	1.34	1.00	1.83	1.83	1.00	2.69	2.69
	mut05-hill	clust-xo-kb-hh	mst-rand-init	1.00	0.53	0.53	1.00	1.48	1.48	1.00	2.05	2.05	1.00	2.83	2.83
	mut05-hill	match-xo	heur-init	1.00	0.54	0.54	1.00	1.31	1.31	1.00	1.84	1.84	1.00	2.73	2.73
	mut05-hill	match-xo	heur-rand-init	1.00	0.55	0.55	1.00	1.40	1.40	1.00	1.97	1.97	1.00	2.70	2.70
	mut05-hill	match-xo	heur-mst-init	1.00	0.55	0.55	1.00	1.48	1.48	1.00	2.01	2.01	1.00	2.84	2.84
	mut05-hill	match-xo	heur-mst-rand-init	1.00	0.55	0.55	1.00	1.51	1.51	1.00	2.13	2.13	1.00	2.88	2.88
	mut05-hill	match-xo	rand-init	1.00	0.51	0.51	1.00	1.38	1.38	1.00	1.92	1.92	1.00	2.94	2.94
	mut05-hill	match-xo	mst-init	1.00	0.47	0.47	1.00	1.36	1.36	1.00	1.96	1.96	1.00	2.95	2.95
	mut05-hill	match-xo	mst-rand-init	1.00	0.52	0.52	1.00	1.49	1.49	1.00	2.05	2.05	1.00	2.99	2.99
	mut05-hill	uform-xo	heur-init	1.00	0.51	0.51	1.00	1.48	1.48	1.00	2.32	2.32	1.00	3.72	3.72
	mut05-hill	uform-xo	heur-rand-init	1.00	0.56	0.56	1.00	1.57	1.57	1.00	2.38	2.38	1.00	3.66	3.66
	mut05-hill	uform-xo	heur-mst-init	1.00	0.53	0.53	1.00	1.67	1.67	1.00	2.44	2.44	1.00	3.81	3.81
	mut05-hill	uform-xo	heur-mst-rand-init	1.00	0.56	0.56	1.00	1.70	1.70	1.00	2.56	2.56	1.00	3.93	3.93
	mut05-hill	uform-xo	rand-init	1.00	0.53	0.53	1.00	1.55	1.55	1.00	2.41	2.41	1.00	4.00	4.00
	mut05-hill	uform-xo	mst-init	1.00	0.46	0.46	1.00	1.41	1.41	1.00	2.30	2.30	1.00	3.77	3.77
	mut05-hill	uform-xo	mst-rand-init	1.00	0.53	0.53	1.00	1.64	1.64	1.00	2.48	2.48	1.00	3.86	3.86

Table A.1: (continued)

Class	Mutation	Crossover	Initialization	100			500			1000			2000		
				SR	**SRT**	**ERT**	**SR**	**SRT**	**ERT**	**SR**	**SRT**	**ERT**	**SR**	**SRT**	**ERT**
S2	mut05-heur	clust-xo	heur-init	1.00	0.47	0.47	1.00	1.34	1.34	1.00	1.87	1.87	1.00	3.47	3.47
	mut05-heur	clust-xo	heur-rand-init	1.00	0.54	0.54	1.00	1.52	1.52	1.00	2.03	2.03	1.00	3.80	3.80
	mut05-heur	clust-xo	heur-mst-init	1.00	0.48	0.48	1.00	1.63	1.63	1.00	2.22	2.22	1.00	4.06	4.06
	mut05-heur	clust-xo	heur-mst-rand-init	1.00	0.50	0.50	1.00	1.75	1.75	1.00	2.39	2.39	1.00	4.26	4.26
	mut05-heur	clust-xo	rand-init	1.00	0.57	0.57	1.00	1.60	1.60	1.00	2.38	2.38	1.00	4.88	4.88
	mut05-heur	clust-xo	mst-init	1.00	0.56	0.57	1.00	1.80	1.80	1.00	2.48	2.48	1.00	4.72	4.72
	mut05-heur	clust-xo	mst-rand-init	1.00	0.56	0.56	1.00	1.89	1.89	1.00	2.78	2.78	1.00	5.07	5.07
	mut05-heur	clust-xo-hh	heur-init	1.00	0.48	0.48	1.00	1.32	1.32	1.00	1.85	1.85	1.00	3.67	3.67
	mut05-heur	clust-xo-hh	heur-rand-init	1.00	0.55	0.55	1.00	1.49	1.49	1.00	2.12	2.12	1.00	4.11	4.11
	mut05-heur	clust-xo-hh	heur-mst-init	1.00	0.50	0.50	1.00	1.70	1.70	1.00	2.33	2.33	1.00	4.64	4.64
	mut05-heur	clust-xo-hh	heur-mst-rand-init	1.00	0.52	0.52	1.00	1.78	1.78	1.00	2.57	2.57	1.00	5.26	5.26
	mut05-heur	clust-xo-hh	rand-init	1.00	0.60	0.60	1.00	1.72	1.72	1.00	2.97	2.97	1.00	7.39	7.39
	mut05-heur	clust-xo-hh	mst-init	1.00	0.59	0.59	1.00	2.65	2.65	1.00	3.38	3.38	1.00	5.86	5.86
	mut05-heur	clust-xo-hh	mst-rand-init	1.00	0.56	0.56	1.00	2.29	2.29	1.00	3.60	3.60	1.00	7.25	7.25
S3	mut05	clust-xo-hh	heur-init	1.00	0.50	0.50	1.00	1.31	1.31	1.00	1.65	1.65	1.00	2.95	2.95
	mut05	clust-xo-hh	heur-rand-init	1.00	0.55	0.55	1.00	1.40	1.40	1.00	1.76	1.76	1.00	3.02	3.02
	mut05	clust-xo-hh	heur-mst-init	1.00	0.48	0.48	1.00	1.48	1.48	1.00	1.83	1.83	1.00	3.16	3.16
	mut05	clust-xo-hh	heur-mst-rand-init	1.00	0.52	0.52	1.00	1.54	1.54	1.00	1.98	2.02	1.00	3.26	3.26
	mut05	clust-xo-hh	rand-init	1.00	0.61	0.61	1.00	1.54	1.54	1.00	2.09	2.09	1.00	3.73	3.73
	mut05	clust-xo-hh	mst-init	0.98	0.44	0.48	1.00	1.46	1.46	1.00	1.77	1.77	1.00	2.96	2.96
	mut05	clust-xo-hh	mst-rand-init	1.00	0.55	0.55	1.00	1.63	1.65	1.00	2.00	2.00	1.00	3.27	3.27
H1	mut05-heur	clust-xo-kb	heur-init	0.45	0.59	3.01	0.43	1.74	11.72	0.25	2.47	45.63	0.11	5.84	262.15
	mut05-heur	clust-xo-kb	heur-rand-init	0.44	0.70	3.32	0.35	1.74	15.88	0.20	2.96	62.61	0.14	4.51	202.21
	mut05-heur	clust-xo-kb	heur-mst-init	0.41	0.59	3.59	0.16	1.88	42.11	0.09	2.33	160.88	0.10	6.64	297.88
	mut05-heur	clust-xo-kb	heur-mst-rand-init	0.42	0.61	3.44	0.18	2.04	38.70	0.10	4.15	143.21	0.03	11.79	1258.01
	mut05-heur	clust-xo-kb	rand-init	0.40	0.82	3.92	0.08	2.20	96.30	0.04	4.66	352.50	0.01	4.16	3350.03
	mut05-heur	clust-xo-kb-hh	heur-init	0.32	0.63	5.00	0.25	1.58	24.92	0.18	2.09	70.05	0.08	4.99	395.84
	mut05-heur	clust-xo-kb-hh	heur-rand-init	0.33	0.73	5.04	0.25	1.89	24.67	0.15	3.08	87.95	0.04	3.73	894.76
	mut05-heur	clust-xo-kb-hh	heur-mst-init	0.33	0.50	4.68	0.11	1.86	67.32	0.07	3.54	198.88	0.07	7.63	437.83
	mut05-heur	clust-xo-kb-hh	heur-mst-rand-init	0.35	0.55	4.49	0.12	2.26	61.77	0.04	2.24	406.59	0.05	9.25	688.81
	mut05-heur	clust-xo-kb-hh	rand-init	0.25	0.92	7.36	0.03	2.93	249.11	0.04	5.74	404.24			
	mut05-heur	match-xo	heur-init	0.02	0.45	119.88	0.10	1.66	68.04	0.04	4.31	327.77			
	mut05-heur	match-xo	heur-rand-init	0.03	0.51	60.25	0.04	2.03	196.40	0.03	2.71	438.88	0.01	14.70	6060.56
	mut05-heur	match-xo	heur-mst-init	0.09	0.33	19.51	0.01	2.03	1411.69						
	mut05-heur	match-xo	heur-mst-rand-init	0.08	0.51	23.61	0.01	3.08	1421.17	0.01	1.67	2671.65			
	mut05-heur	match-xo	rand-init	0.03	0.52	61.67	0.11	2.58	60.57	0.05	3.66	264.93	0.01	29.96	6452.60

Table A.1: (continued)

				100			500			1000			2000		
Class	Mutation	Crossover	Initialization	**SR**	**SRT**	**ERT**	**SR**	**SRT**	**ERT**	**SR**	**SRT**	**ERT**	**SR**	**SRT**	**ERT**
H2	mut05-heur	clust-xo-kb	mst-init	0.20	0.47	9.14									
	mut05-heur	clust-xo-kb-hh	mst-init	0.21	0.49	8.74									
	mut05-heur	match-xo	mst-init	0.11	0.39	15.92									
	mut05-heur	uform-xo	mst-init	0.12	0.46	13.90									
	mut05-heur	clust-xo-kb	mst-rand-init	0.40	0.65	3.84	0.01	1.48	1525.70				0.01	9.42	6506.98
	mut05-heur	clust-xo-kb-hh	mst-rand-init	0.30	0.56	5.52	0.01	1.72	1510.03				0.01	16.81	6491.14
	mut05-heur	match-xo	mst-rand-init	0.10	0.43	18.67									
	mut05-heur	uform-xo	mst-rand-init	0.10	0.53	16.30									
H3	mut05-heur	uform-xo	heur-init	0.12	0.70	12.75	0.13	1.94	48.68	0.12	2.18	102.36	0.10	4.47	296.31
	mut05-heur	uform-xo	heur-rand-init	0.13	0.79	12.16	0.12	2.02	54.06	0.10	2.42	125.51	0.10	5.14	285.13
	mut05-heur	uform-xo	heur-mst-init	0.11	0.50	14.47	0.02	2.47	453.50	0.04	3.67	379.52	0.02	4.25	2055.31
	mut05-heur	uform-xo	heur-mst-rand-init	0.08	0.58	21.78	0.02	1.85	341.53	0.04	4.13	378.13	0.04	7.93	844.42
	mut05-heur	uform-xo	rand-init	0.14	0.79	11.61	0.20	2.18	31.93	0.16	2.45	78.62	0.10	4.73	316.75
M1	mut05	match-xo	heur-init	0.31	1.33	4.57	0.24	42.36	216.27	0.27	198.67	769.48			
	mut05	match-xo	heur-rand-init	0.39	1.62	4.30	0.25	41.40	210.32	0.26	216.84	782.76			
	mut05	match-xo	heur-mst-init	0.29	0.76	3.68	0.28	42.09	182.09	0.25	211.66	906.66			
	mut05	match-xo	heur-mst-rand-init	0.21	0.80	5.42	0.22	42.26	237.62	0.21	220.91	1023.79	0.01	895.36	3891.60
	mut05	match-xo	rand-init	0.46	1.99	4.41	0.40	52.11	155.99	0.38	259.86	834.90	0.42	1329.69	3919.01
	mut05	match-xo	mst-rand-init	0.25	0.74	4.27									
	mut05	uform-xo	heur-init	0.46	1.09	2.61	0.28	38.39	172.49	0.24	190.32	822.53			
	mut05	uform-xo	heur-rand-init	0.43	1.13	2.96	0.25	39.64	208.62	0.27	193.43	752.35	0.01	581.40	3598.08
	mut05	uform-xo	heur-mst-init	0.32	0.75	2.87	0.26	39.30	190.55	0.30	215.79	678.44	0.01	881.09	2617.79
	mut05	uform-xo	heur-mst-rand-init	0.24	0.76	3.93	0.21	36.18	236.72	0.22	204.64	1007.08			
	mut05	uform-xo	rand-init	0.55	1.40	2.93	0.33	50.62	189.69	0.41	251.28	743.40	0.45	1329.61	3662.32
	mut05	uform-xo	mst-rand-init	0.26	0.63	3.37	0.10	52.12	185.39	0.01	231.01	1311.49			
M2	mut05	uform-xo	mst-init	0.25	0.53	3.26									
M3	mut05	match-xo	mst-init	0.27	0.64	3.55									
M4	mut05	clust-xo	heur-init	0.01	2.13	318.48									
	mut05	clust-xo	heur-rand-init	0.01	0.45	319.76									
	mut05	clust-xo	heur-mst-init	0.07	0.53	22.17									
	mut05	clust-xo	heur-mst-rand-init	0.04	0.34	45.30									
	mut05	clust-xo	rand-init												
	mut05	clust-xo	mst-init	0.11	0.74	14.40									
	mut05	clust-xo	mst-rand-init	0.06	0.71	27.94									
M5	mut05	clust-xo-kb	heur-init	0.06	0.92	26.13									
	mut05	clust-xo-kb	heur-rand-init	0.01	0.26	301.65									
	mut05	clust-xo-kb	heur-mst-init	0.12	0.63	11.83									
	mut05	clust-xo-kb	heur-mst-rand-init	0.04	0.35	42.76									
	mut05	clust-xo-kb	rand-init												

Table A.1: (continued)

				100			500			1000			2000		
Class	Mutation	Crossover	Initialization	**SR**	**SRT**	**ERT**	**SR**	**SRT**	**ERT**	**SR**	**SRT**	**ERT**	**SR**	**SRT**	**ERT**
M6	mut05	clust-xo-kb-hh	heur-init	0.39	0.70	3.12	0.38	2.30	10.06	0.01	1.83	1331.25			
	mut05	clust-xo-kb-hh	heur-rand-init	0.38	0.83	3.43	0.15	2.42	31.06						
	mut05	clust-xo-kb-hh	heur-mst-init	0.46	0.62	2.53	0.17	2.71	27.45						
	mut05	clust-xo-kb-hh	heur-mst-rand-init	0.43	0.68	2.88	0.08	2.83	62.91						
	mut05	clust-xo-kb-hh	rand-init	0.03	1.32	63.61									
M7	mut05	clust-xo-kb	mst-init	0.08	0.30	18.31									
	mut05	clust-xo-kb	mst-rand-init	0.05	0.35	28.77									
	mut05	clust-xo-kb-hh	mst-init	0.24	0.51	5.43									
	mut05	clust-xo-kb-hh	mst-rand-init	0.33	0.74	4.08									

A.2 Results for Successful Configurations and Large Problems

In Section 4.4.2 we have selected some of the successful algorithm configurations for further evaluation. We have evaluated them up to problem sizes of 100,000 objects. This section presents the detailed measurement results, average over 200 runs as above.

For each tested configuration we show the measured CPU time, split up into initialization time and search time, the average number of generations until success, and the average time spent to process a single generation.

Table A.2: Evaluation of algorithms using cluster-based crossover with a heuristic homeless strategy

	clust-xo-hh, mut05, heur-init								
Problem Size	500	1,000	2,000	5,000	10,000	25,000	50,000	75,000	100,000
Init. Time	3.3	3.7	4.8	4.2	5.7	20.5	41.3	66.7	97.4
Search Time	2.4	2.9	3.4	9.5	24.5	117.7	319.7	577.7	892.2
Total Time	5.7	6.6	8.2	13.7	30.2	138.2	361.0	644.4	989.6
Generations	10.1	12.0	14.4	18.0	21.5	27.1	31.5	34.5	37.1
Time / Gen.	0.2	0.2	0.2	0.5	1.1	4.3	10.2	16.7	24.1

	clust-xo-hh, mut05, mst-init								
Problem Size	500	1,000	2,000	5,000	10,000	25,000	50,000	75,000	100,000
Init. Time	2.7	3.1	3.9	3.4	3.6	11.5	17.6	28.1	38.8
Search Time	3.5	3.9	4.7	9.6	24.0	111.7	309.7	561.9	893.4
Total Time	6.1	7.0	8.6	13.0	27.7	123.3	327.3	590.0	932.2
Generations	13.3	12.5	14.3	17.1	20.0	24.9	29.8	33.0	35.5
Time / Gen.	0.3	0.3	0.3	0.6	1.2	4.5	10.4	17.0	25.1

Table A.3: Evaluation of algorithms using the hill-climbing heuristic

	clust-xo-kb-hh, mut05-hill, heur-init								
Problem Size	500	1,000	2,000	5,000	10,000	25,000	50,000	75,000	100,000
Init. Time	3.2	3.9	5.0	4.2	5.6	20.3	41.4	66.4	94.7
Search Time	2.3	3.0	3.2	7.2	18.2	80.9	236.6	386.0	585.9
Total Time	5.6	6.9	8.2	11.5	23.9	101.3	278.0	452.3	680.6
Generations	2.0	2.8	3.0	3.7	4.0	5.0	5.9	6.0	6.2
Time / Gen.	1.1	1.1	1.1	2.0	4.6	16.2	40.0	64.3	94.3

	clust-xo-kb-hh, mut05-hill, heur-rand-init								
Problem Size	500	1,000	2,000	5,000	10,000	25,000	50,000	75,000	100,000
Init. Time	3.2	4.1	4.9	4.4	4.8	16.5	28.6	46.4	65.7
Search Time	2.4	3.1	3.3	7.7	17.8	81.9	242.1	390.0	627.7
Total Time	5.6	7.2	8.2	12.1	22.6	98.4	270.7	436.4	693.4
Generations	2.2	2.9	3.0	4.0	4.0	5.0	6.0	6.0	6.5
Time / Gen.	1.1	1.0	1.1	1.9	4.4	16.4	40.5	64.8	96.8

	clust-xo-kb-hh, mut05-hill, mst-init								
Problem Size	500	1,000	2,000	5,000	10,000	25,000	50,000	75,000	100,000
Init. Time	2.5	3.2	3.9	3.3	3.7	11.4	17.7	28.3	38.0
Search Time	3.3	4.0	4.6	8.6	21.5	81.7	216.3	370.9	548.0
Total Time	5.9	7.2	8.5	11.9	25.2	93.1	234.0	399.2	585.9
Generations	2.9	3.1	3.9	4.4	5.0	5.2	5.7	5.9	6.0
Time / Gen.	1.1	1.3	1.2	1.9	4.3	15.7	38.1	62.4	91.3

	match-xo, mut05-hill, heur-init algorithm								
Problem Size	500	1,000	2,000	5,000	10,000	25,000	50,000	75,000	100,000
Init. Time	3.3	3.8	4.7	4.2	5.7	20.4	41.3	66.0	95.0
Search Time	2.6	3.0	3.3	8.1	20.4	97.0	267.4	466.7	736.7
Total Time	5.9	6.8	8.1	12.3	26.1	117.4	308.7	532.6	831.7
Generations	3.0	3.3	4.0	5.0	5.6	6.7	7.7	8.2	8.9
Time / Gen.	0.8	0.9	0.8	1.6	3.7	14.4	34.7	56.65	82.5

B Experimental Results for Chapter 5

B.1 The Object-grouping Multi-level EA

Table B.1: Detailed average runtime and generations that were spent by the successful object-grouping multi-level EAs. The shown configurations achieved 100% success, the clust-xo-hh, mut05, mst-init configuration had a low SR and is not presented here.

	clust-xo-hh, mut05, heur-init				
Problem Size	10,000	25,000	50,000	75,000	100,000
1st-level runtime	4.1	10.5	22.1	34.3	46.1
2nd-level runtime	24.2	84.4	215.0	378.2	561.9
Total runtime	28.3	94.9	237.1	412.5	608.0
1st-level generations	2.0	2.0	2.0	2.0	2.0
2nd-level generations	11.0	13.6	15.6	16.8	17.7
Total generations	13.0	15.6	17.6	18.8	19.7

	clust-xo-kb-hh, mut05-hill, heur-init				
Problem Size	10,000	25,000	50,000	75,000	100,000
1st-level runtime	5.1	12.3	29.0	48.2	66.1
2nd-level runtime	10.7	26.2	59.2	98.8	137.6
Total runtime	15.8	38.5	88.2	147.0	203.7
1st-level generations	2.0	2.0	2.0	2.0	2.0
2nd-level generations	1.0	1.0	1.0	1.0	1.0
Total generations	3.0	3.0	3.0	3.0	3.0

	clust-xo-kb-hh, mut05-hill, mst-init				
Problem Size	10,000	25,000	50,000	75,000	100,000
1st-level runtime	4.9	10.5	20.9	31.9	41.4
2nd-level runtime	11.1	30.4	89.3	167.5	243.5
Total runtime	16.0	40.9	110.2	199.4	284.9
1st-level generations	1.0	1.0	1.0	1.0	1.0
2nd-level generations	1.0	1.2	1.7	1.9	2.0
Total generations	2.0	2.2	2.7	2.9	3.0

B.2 The Cutting Multi-level EA

Table B.2: Detailed average runtime and generations that were spent by the cutting multi-level EAs. All configurations had a success rate > 99%.

	clust-xo-hh. mut05. heur-init				
Problem Size	10,000	25,000	50,000	75,000	100,000
1st-level runtime	24.6	62.6	137.5	220.2	324.9
2nd-level runtime	35.4	142.5	385.1	706.7	1091
Total runtime	60	205.1	522.6	926.9	1415.9
1st-level generations	10.0	10.0	10.0	10.0	10.0
2nd-level generations	16.7	22.9	27.7	30.9	32.9
Total generations	26.7	32.9	37.7	40.9	42.9

	clust-xo-hh. mut05. mst-init				
Problem Size	10,000	25,000	50,000	75,000	100,000
1st-level runtime	23.2	58.4	124.9	205.9	301.5
2nd-level runtime	35.9	144.3	386.8	721.4	1119.9
Total runtime	59.1	202.7	511.7	927.3	1421.4
1st-level generations	10.0	10.0	10.0	10.0	10.0
2nd-level generations	16.8	23.2	28.2	31.5	33.6
Total generations	26.8	33.2	38.2	41.5	43.6

	clust-xo-kb-hh, mut05-hill, heur-init				
Problem Size	10,000	25,000	50,000	75,000	100,000
1st-level runtime	25.0	61.4	130	218.4	322.3
2nd-level runtime	19.5	74.5	218.2	437.6	671.4
Total runtime	44.5	135.9	348.2	656	993.7
1st-level generations	3.0	3.0	3.0	3.0	3.0
2nd-level generations	2.0	3.0	4.0	4.8	5.0
Total generations	5.0	6.0	7.0	7.8	8.0

	clust-xo-kb-hh, mut05-hill, mst-init				
Problem Size	10,000	25,000	50,000	75,000	100,000
1st-level runtime	22.9	53	112.1	184.3	275.5
2nd-level runtime	25.6	87.6	215.1	394.9	646.6
Total runtime	48.5	140.6	327.2	579.2	922.1
1st-level generations	3.0	3.0	3.0	3.0	3.0
2nd-level generations	3.1	3.8	4.1	4.4	4.9
Total generations	6.1	6.8	7.1	7.4	7.9

Bibliography

[1] E. Alba and M. Tomassini. Parallelism and evolutionary algorithms. *IEEE Transactions on Evolutionary Computation*, 6(5):443–462, 2002. doi:10.1109/TEVC.2002.800880.

[2] E. Alba and B. Dorronsoro. *Cellular Genetic Algorithms*, volume 42 of *Operations Research/Computer Science Interfaces Series*. Springer, 2008. doi:10.1007/978-0-387-77610-1.

[3] V. S. Alves, R. J. G. B. Campello, and E. R. Hruschka. Towards a fast evolutionary algorithm for clustering. In *Proceedings of the 2006 IEEE Congress on Evolutionary Computation (CEC)*, pages 1776–1783. IEEE, 2006. doi:10.1109/CEC.2006.1688522.

[4] G. M. Amdahl. Validity of the single processor approach to achieving large scale computing capabilities. In *Proceedings of the 1967 AFIPS Joint Computer Conferences*, pages 483–485. ACM, 1967. doi:10.1145/1465482.1465560.

[5] S. Bandyopadhyay, H. Kargupta, and G. Wang. Revisiting the GEMGA: scalable evolutionary optimization through linkage learning. In *Proceedings of the 1998 IEEE International Conference on Evolutionary Computation*, pages 603–608. IEEE, 1998. doi:10.1109/ICEC.1998.700097.

[6] J. C. Bezdek, S. Boggavarapu, L. O. Hall, and A. Bensaid. Genetic algorithm guided clustering. In *Proceedings of the First IEEE Conference on Evolutionary Computation*, volume 1, pages 34–39. IEEE, 1994. doi:10.1109/ICEC.1994.350046.

[7] J. Branke, H. C. Andersen, and H. Schmeck. Global selection methods for massively parallel computers. In *Proceedings of the AISB Workshop on Evolutionary Computing*, volume 1143 of *Lecture Notes in Computer Science*, pages 175–188. Springer, 1996. doi:10.1007/BFb0032782.

[8] P. Brucker. On the complexity of clustering problems. In *Optimization and Operations Research: Proceedings of a Workshop Held at the University of Bonn*, volume 157 of *Lecture Notes in Economics and Mathematical Systems*, pages 45–54. Springer, 1978.

[9] E. Burke, G. Kendall, J. Newall, E. Hart, P. Ross, and S. Schulenburg. Hyper-heuristics: An emerging direction in modern search technology. In *Handbook of Metaheuristics*, volume 57 of *International Series in Operations Research & Management Science*, pages 457–474. Springer, 2003. doi:10.1007/0-306-48056-5_16.

[10] E. K. Burke, M. R. Hyde, and G. Kendall. Evolving bin packing heuristics with genetic programming. In *Proceedings of the 9th International Conference on Parallel Problem Solving from Nature (PPSN)*, volume 4193 of *Lecture Notes in Computer Science*, pages 860–869. Springer, 2006. doi:10.1007/11844297_87.

[11] E. Cantú-Paz. A survey of parallel genetic algorithms. *Calculateurs Parallèles, Réseaux et Systèmes Répartis*, 10(2):141–171, 1998.

[12] P. M. S. Carvalho, L. A. F. M. Ferreira, and L. M. F. Barruncho. On spanning-tree recombination in evolutionary large-scale network problems – application to electrical distribution planning. *IEEE Transactions on Evolutionary Computation*, 5(6):623–630, 2001. doi:10.1109/4235.974844.

[13] E. Chen and F. Wang. Dynamic clustering using multi-objective evolutionary algorithm. In *Proceedings of the 2005 International Conference on Computational Intelligence and Security (CIS)*, volume 3801 of *Lecture Notes in Computer Science*, pages 73–80. Springer, 2005. doi:10.1007/11596448_10.

[14] Y.-p. Chen and D. Goldberg. Convergence time for the linkage learning genetic algorithm. *Evolutionary Computation*, 13(3):279–302, 2005. doi:10.1162/1063656054794806.

[15] Y.-p. Chen, T.-L. Yu, K. Sastry, and D. E. Goldberg. A survey of genetic linkage learning techniques. Technical Report 2007014, Illinois Genetic Algorithms Laboratory, University of Illinois at Urbana-Champaign, 2007. Available from: http://www.illigal.uiuc.edu/pub/papers/IlliGALs/2007014.pdf.

[16] S.-S. Choi and B.-R. Moon. Normalization in genetic algorithms. In *Proceedings of the 2003 Genetic and Evolutionary Computation Conference (GECCO)*, volume 2723 of *Lecture Notes in Computer Science*, pages 862–873. Springer, 2003. doi:10.1007/3-540-45105-6_99.

[17] I. T. Christou, A. Zakarian, J.-M. Liu, and H. Carter. A two-phase genetic algorithm for large-scale bidline-generation problems at Delta Air Lines. *INTERFACES*, 29(5):51–65, 1999. doi:10.1287/inte.29.5.51.

[18] R. M. Cole. Clustering with genetic algorithms. Master's thesis, University of Western Australia, 1998. Available from: http://www.csse.uwa.edu.au/pub/robvis/theses/RowenaCole.ps.gz.

[19] W. J. Conover. *Practical Nonparametric Statistics*. Wiley, 1980.

[20] D. W. Corne, N. R. Jerram, J. D. Knowles, and M. J. Oates. PESA-II: Region-based selection in evolutionary multiobjective optimization. In *Proceedings of the 2001 Genetic and Evolutionary Computation Conference (GECCO)*, pages 283–290. Morgan Kaufmann, 2001.

[21] P. Cowling, G. Kendall, and L. Han. An investigation of a hyperheuristic genetic algorithm applied to a trainer scheduling problem. In *Proceedings of the 2002 IEEE Congress on Evolutionary Computation (CEC)*, volume 2, pages 1185–1190. IEEE, 2002. doi:10.1109/CEC.2002.1004411.

[22] K. Deb, S. Agrawal, A. Pratap, and T. Meyarivan. A fast elitist non-dominated sorting genetic algorithm for multi-objective optimization: NSGA-II. In *Proceedings of the 6th International Conference on Parallel Problem Solving from Nature (PPSN)*, volume 1917 of *Lecture Notes in Computer Science*, pages 849–858. Springer, 2000. doi:10.1007/3-540-45356-3_83.

[23] K. Deb, A. R. Reddy, and G. Singh. Optimal scheduling of casting sequence using genetic algorithms. *Materials and Manufacturing Processes*, 18(3):409–432, 2003. doi:10.1081/AMP-120022019.

[24] G. N. Demir, A. Ş. Uyar, and Ş. Gündüz-Öğüdücü. Multiobjective evolutionary clustering of web user sessions: A case study in web page recommendation *Soft Computing*, 2009. (forthcoming). doi:10.1007/s00500-009-0428-y.

[25] A. G. Di Nuovo, M. Palesi, and V. Catania. Multi-objective evolutionary fuzzy clustering for high-dimensional problems. In *Proceedings of the 2007 IEEE International Fuzzy Systems Conference (FUZZ-IEEE)*, pages 1–6. IEEE, 2007. doi:10.1109/FUZZY.2007.4295660.

[26] C. R. Dias and L. S. Ochi. Efficient evolutionary algorithms for the clustering problem in directed graphs. In *Proceedings of the 2003 IEEE Congress on Evolutionary Computation (CEC)*, volume 2, pages 983–990. IEEE, 2003. doi:10.1109/CEC.2003.1299774.

[27] J. Du, E. Korkmaz, R. Alhajj, and K. Barker. Novel clustering approach that employs genetic algorithm with new representation scheme and multiple objectives. In *Proceedings of the 6th International Conference on Data Warehousing and Knowledge Discovery (DaWaK)*, volume 3181 of *Lecture Notes in Computer Science*, pages 219–228. Springer, 2004. doi:10.1007/b99817.

[28] A. E. Eiben and J. E. Smith. *Introduction to Evolutionary Computing.* Natural Computing Series. Springer, 2nd edition, 2007. Available from: http://books.google.com/books?id=7IOE5VIpFpwC.

[29] E. Falkenauer. *Genetic Algorithms and Grouping Problems.* Wiley, 1998.

[30] D. B. Fogel. Introduction. In T. Bäck, D. B. Fogel, and Z. Michalewicz, editors, *Handbook of Evolutionary Computation*, page A1.1. Oxford University Press and Institute of Physics, 1997. Available from: http://books.google.com/books?id=n5nuiIZvmpAC.

[31] L. J. Fogel, A. J. Owens, and M. J. Walsh. *Artificial intelligence through simulated evolution*. Wiley, 1966.

[32] G. Gan, C. Ma, and J. Wu. *Data Clustering: Theory, Algorithms, and Applications*. Series on Statistics and Applied Probability. Society for Industrial and Applied Mathematics, 2007.

[33] J. Gasvoda and Q. Ding. A genetic algorithm for clustering on very large data sets. In *Proceedings of the 16th International Conference on Computer Applications in Industry and Engineering (CAINE)*, pages 163–167. International Society for Computers and Their Applications (ISCA), 2003.

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

[35] D. E. Goldberg, B. Korb, and K. Deb. Messy genetic algorithms: Motivation, analysis, and first results. *Complex Systems*, 3(5):493–530, 1989.

[36] J. C. Gower and G. J. S. Ross. Minimum spanning trees and single linkage cluster analysis. *Applied Statistics*, 18(1):54–64, 1969. Available from: `http://www.jstor.org/stable/2346439`.

[37] Ş. Gündüz-Öğüdücü and A. Ş. Uyar. A graph based clustering method using a hybrid evolutionary algorithm *WSEAS Transactions on Mathematics*, 3(3):731–736, 2004.

[38] J. Handl and J. Knowles. Exploiting the trade-off – the benefits of multiple objectives in data clustering. In *Proceedings of the 3rd International Conference on Evolutionary Multi-Criterion Optimization (EMO)*, volume 3410 of *Lecture Notes in Computer Science*, pages 547–560. Springer, 2005. `doi:10.1007/b106458`.

[39] J. Handl and J. Knowles. An evolutionary approach to multiobjective clustering. *IEEE Transactions on Evolutionary Computation*, 11(1):56–76, 2007. `doi:10.1109/TEVC.2006.877146`.

[40] G. R. Harik, F. G. Lobo, and K. Sastry. Linkage learning via probabilistic modeling in the extended compact genetic algorithm (ECGA). In *Scalable Optimization via Probabilistic Modeling*, volume 33 of *Studies in Computational Intelligence*, pages 39–61. Springer, 2006. `doi:10.1007/978-3-540-34954-9_3`.

[41] W. E. Hart, N. Krasnogor, and J. Smith, editors. *Recent Advances in Memetic Algorithms*, volume 166 of *Studies in Fuzziness and Soft Computing*. Springer, 2005. `doi:10.1007/3-540-32363-5`.

[42] P. R. Hinton. *Statistics explained*. Routledge, 2nd edition, 2004. Available from: `http://books.google.com/books?id=16PzAY8g0F4C`.

[43] J. H. Holland. *Adaptation in Natural and Artificial Systems*. University of Michigan Press, 1975.

[44] J. Homberger and H. Gehring. A two-phase hybrid metaheuristic for the vehicle routing problem with time windows. *European Journal of Operational Research*, 162(1):220–238, 2005. Logistics: From Theory to Application. doi:10.1016/j.ejor.2004.01.027.

[45] J. Horn, N. Nafpliotis, and D. Goldberg. A niched pareto genetic algorithm for multiobjective optimization. In *Proceedings of the First IEEE Conference on Evolutionary Computation*, volume 1, pages 82–87. IEEE, 1994. doi:10.1109/ICEC.1994.350037.

[46] E. R. Hruschka, R. J. G. B. Campello, and L. N. de Castro. Evolving clusters in gene-expression data. *Information Sciences*, 176(13):1898–1927, 2006. doi:10.1016/j.ins.2005.07.015.

[47] E. R. Hruschka, R. J. G. B. Campello, A. A. Freitas, and A. C. P. L. F. de Carvalho. A survey of evolutionary algorithms for clustering. *IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews*, 39(2):133–155, 2009. doi:10.1109/TSMCC.2008.2007252.

[48] L. Hubert and P. Arabie. Comparing partitions. *Journal of Classification*, 2(1):193–218, 1985. doi:10.1007/BF01908075.

[49] A. K. Jain, M. N. Murty, and P. J. Flynn. Data clustering: a review. *ACM Computing Surveys*, 31(3):264–323, 1999. doi:10.1145/331499.331504.

[50] L. Jie, G. Xinbo, and J. Li-cheng. A GA-based clustering algorithm for large data sets with mixed and categorical values. In *Proceedings of the 5th International Conference on Computational Intelligence and Multimedia Applications (ICCIMA)*, pages 102–107. IEEE, 2003. Available from: http://ieeexplore.ieee.org/search/wrapper.jsp?arnumber=1238108.

[51] H. Kargupta and S. Bandyopadhyay. A perspective on the foundation and evolution of the linkage learning genetic algorithms. *Computer Methods in Applied Mechanics and Engineering*, 186(2):269–294, 2000. doi:10.1016/S0045-7825(99)00387-4.

[52] J. R. Kettenring. The practice of cluster analysis. *Journal of Classification*, 23(1):3–30, 2006. doi:10.1007/s00357-006-0002-6.

[53] Y.-H. Kim, Y. Yoon, A. Moraglio, and B.-R. Moon. Geometric crossover for multiway graph partitioning. In *Proceedings of the 2006 Genetic and Evolutionary Computation Conference (GECCO)*, pages 1217–1224. ACM, 2006. doi:10.1145/1143997.1144189.

[54] E. E. Korkmaz. A two-level clustering method using linear linkage encoding. In *Proceedings of the 9th International Conference on Parallel Problem Solving from Nature (PPSN)*, volume 4193 of *Lecture Notes in Computer Science*, pages 681–690. Springer, 2006. doi:10.1007/11844297_69.

[55] J. R. Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection.* MIT Press, 1992. Available from: http://books.google.com/books?id=Bhtxo60BV0EC.

[56] R. Krovi. Genetic algorithms for clustering: a preliminary investigation. In *Proceedings of the 25th Hawaii International Conference on System Sciences (HICSS)*, volume 4, pages 540–544. IEEE, 1992. doi:10.1109/HICSS.1992.183445.

[57] R. Kumar, A. H. Joshi, K. K. Banka, and P. I. Rockett. Evolution of hyperheuristics for the biobjective 0/1 knapsack problem by multiobjective genetic programming. In *Proceedings of the 2008 Genetic and Evolutionary Computation Conference (GECCO)*, pages 1227–1234. ACM, 2008. doi:10.1145/1389095.1389335.

[58] G. von Laszewski. Intelligent structural operators for the k-way graph partitioning problem In *Proceedings of the 4th International Conference on Genetic Algorithms (ICGA)*, pages 45–52. Morgan Kaufmann, 1991.

[59] E. Leon, O. Nasraoui, and J. Gomez. ECSAGO: Evolutionary clustering with self adaptive genetic operators. In *Proceedings of the 2006 IEEE Congress on Evolutionary Computation (CEC)*, pages 1768–1775. IEEE, 2006. doi:10.1109/CEC.2006.1688521.

[60] C. B. Lucasius, A. D. Dane, and G. Kateman. On k-medoid clustering of large data sets with the aid of a genetic algorithm: background, feasiblity and comparison. *Analytica Chimica Acta*, 282(3):647–669, 1993. doi:10.1016/0003-2670(93)80130-D.

[61] W. N. Martin, J. Lienig, and J. P. Cohoon. Island (migration) models: evolutionary algorithms based on punctuated equilibria. In T. Bäck, D. B. Fogel, and Z. Michalewicz, editors, *Handbook of Evolutionary Computation*, page C6.3. Oxford University Press and Institute of Physics, 1997. Available from: http://books.google.com/books?id=n5nuiIZvmpAC.

[62] F. Martino and A. Spoto. Social network analysis: A brief theoretical review and further perspectives in the study of information technology. *PsychNology*, 4(1):53–86, 2006. Available from: http://psychnology.org/File/PNJ4(1)/PSYCHNOLOGY_JOURNAL_4_1_MARTINO.pdf.

[63] P. Merz and A. Zell. Clustering gene expression profiles with memetic algorithms. In *Proceedings of the 7th International Conference on Parallel Problem Solving from Nature (PPSN)*, volume 2439 of *Lecture Notes in Computer Science*, pages 811–820. Springer, 2002. doi:10.1007/3-540-45712-7_78.

[64] H. Mühlenbein. Parallel genetic algorithms in combinatorial optimization. In O. Balci, R. Sharda, and S. A. Zenios, editors, *Computer Science and Operations Research: New Developments in Their Interfaces*. Pergamon Press, 1992.

[65] A. Mukhopadhyay and U. Maulik. Multiobjective approach to categorical data clustering. In *Proceedings of the 2007 IEEE Congress on Evolutionary Computation (CEC)*, pages 1296–1303. IEEE, 2007. `doi:10.1109/CEC.2007.4424620`.

[66] M. Munetomo and D. E. Goldberg. Linkage identification by non-monotonicity detection for overlapping functions. *Evolutionary Computation*, 7(4):377–398, 1999. `doi:10.1162/evco.1999.7.4.377`.

[67] D. R. Nadeau. Java tip: How to get cpu, system, and user time for benchmarking. Nadeau Software Consulting, 2008. Retrieved from `http://nadeausoftware.com/articles/2008/03/java_tip_how_get_cpu_and_user_time_benchmarking` on March 11, 2009.

[68] M. C. Naldi, A. C. P. L. F. de Carvalho, R. J. G. B. Campello, and E. R. Hruschka. Genetic clustering for data mining. In O. Maimon and L. Rokach, editors, *Soft Computing for Knowledge Discovery and Data Mining*, pages 113–132. Springer, 2007. `doi:10.1007/978-0-387-69935-6_5`.

[69] Y.-S. Ong, M.-H. Lim, F. Neri, and H. Ishibuchi. Special issue on emerging trends in soft computing: memetic algorithms. *Soft Computing*, 13(8-9):739–917, 2009. `doi:10.1007/s00500-008-0353-5`.

[70] S. Paterlini and T. Krink. High performance clustering with differential evolution. In *Proceedings of the 2004 IEEE Congress on Evolutionary Computation (CEC)*, volume 2, pages 2004–2011. IEEE, 2004. Available from: `http://ieeexplore.ieee.org/search/wrapper.jsp?arnumber=1331142`.

[71] M. Pelikan, K. Sastry, and D. E. Goldberg. Sporadic model building for efficiency enhancement of the hierarchical BOA. *Genetic Programming and Evolvable Machines*, 9(1):53–84, 2008. `doi:10.1007/s10710-007-9052-8`.

[72] V. Rayward-Smith. Metaheuristics for clustering in KDD. In *Proceedings of the 2005 IEEE Congress on Evolutionary Computation (CEC)*, volume 3, pages 2380–2387. IEEE, 2005. `doi:10.1109/CEC.2005.1554991`.

[73] I. Rechenberg. *Evolutionsstrategie: Optimierung technischer Systeme nach Prinzipien der biologischen Evolution*. Frommann-Holzboog, 1973.

[74] H. Schmeck, J. Branke, and U. Kohlmorgen. Parallel implementations of evolutionary algorithms. In A. Y. Zomaya, F. Ercal, and S. Olariu, editors, *Solutions to Parallel and Distributed Computing Problems: Lessons from Biological Sciences*, pages 47–66. Wiley, 2001.

[75] G. P. Scott, D. I. Clark, and T. Pham. A genetic clustering algorithm guided by a descent algorithm. In *Proceedings of the 2001 IEEE Congress on Evolutionary Computation (CEC)*, volume 2, pages 734–740. IEEE, 2001. doi:10.1109/CEC.2001.934262.

[76] D.-I. Seo and B.-R. Moon. A survey on chromosomal structures and operators for exploiting topological linkages of genes. In *Proceedings of the 2003 Genetic and Evolutionary Computation Conference (GECCO)*, volume 2724 of *Lecture Notes in Computer Science*, pages 1357–1368. Springer, 2003. doi:10.1007/3-540-45110-2_10.

[77] W. Sheng and X. Liu. A hybrid algorithm for k-medoid clustering of large data sets. In *Proceedings of the 2004 IEEE Congress on Evolutionary Computation (CEC)*, volume 1, pages 77–82. IEEE, 2004. doi:10.1109/CEC.2004.1330840.

[78] A. da Silva, Y. Lechevallier, F. Rossi, and F. de Carvalho. Clustering dynamic web usage data In N. Nedjah, L. de Macedo Mourelle, and J. Kacprzyk, editors, *Innovative Applications in Data Mining*, volume 169 of *Studies in Computational Intelligence*, pages 71–82. Springer, 2009. doi:10.1007/978-3-540-88045-5_4.

[79] N. Speer, P. Merz, C. Spieth, and A. Zell. Clustering gene expression data with memetic algorithms based on minimum spanning trees. In *Proceedings of the 2003 IEEE Congress on Evolutionary Computation (CEC)*, volume 3, pages 1848–1855. IEEE, 2003. doi:10.1109/CEC.2003.1299897.

[80] C. Stephens and H. Waelbroeck. Schemata evolution and building blocks. *Evolutionary Computation*, 7(2):109–124, 1999. doi:10.1162/evco.1999.7.2.109.

[81] P. D. Surry and N. J. Radcliffe. Inoculation to initialise evolutionary search. In *Proceedings of the AISB Workshop on Evolutionary Computing*, volume 1143 of *Lecture Notes in Computer Science*, pages 269–285. Springer, 1996. doi:10.1007/BFb0032789.

[82] D. Tasoulis and M. Vrahatis. The new window density function for efficient evolutionary unsupervised clustering. In *Proceedings of the 2005 IEEE Congress on Evolutionary Computation (CEC)*, volume 3, pages 2388–2394. IEEE, 2005. doi:10.1109/CEC.2005.1554992.

[83] A. Ş. Uyar and Ş. Gündüz-Öğüdücü. A new graph-based evolutionary approach to sequence clustering In *Proceedings of the 2005 International Conference on Machine Learning and Applications (ICMLA)*, pages 273–278. IEEE, 2005. doi:10.1109/ICMLA.2005.4.

[84] M. D. Vose and A. H. Wright. Form invariance and implicit parallelism. *Evolutionary Computation*, 9(3):355–370, 2001. doi:10.1162/106365601750406037.

[85] D. H. Wolpert and W. G. Macready. No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation*, 1(1):67–82, 1997. doi:10.1109/4235.585893.

[86] A. H. Wright, M. D. Vose, and J. E. Rowe. Implicit parallelism. In *Proceedings of the 2003 Genetic and Evolutionary Computation Conference (GECCO)*, volume 2724 of *Lecture Notes in Computer Science*, pages 1505–1517. Springer, 2003. doi:10.1007/3-540-45110-2_22.

[87] J. Xiao, Y. Yan, Y. Lin, L. Yuan, and J. Zhang. A quantum-inspired genetic algorithm for data clustering. In *Proceedings of the 2008 IEEE Congress on Evolutionary Computation (CEC)*, pages 1513–1519. IEEE, 2008. doi:10.1109/CEC.2008.4630993.

[88] R. Xu and D. Wunsch II. Survey of clustering algorithms. *IEEE Transactions on Neural Networks*, 16(3):645–678, 2005. doi:10.1109/TNN.2005.845141.

[89] R. Xu and D. C. Wunsch II. *Clustering*. IEEE Press Series on Computational Intelligence. Wiley-IEEE Press, 2008.

[90] E. Zitzler, M. Laumanns, and L. Thiele. SPEA2: Improving the strength pareto evolutionary algorithm. Technical Report 103, Computer Engineering and Networks Laboratory, ETH Zürich, 2001. Available from: ftp://ftp.tik.ee.ethz.ch/pub/people/zitzler/ZLT2001a.pdf.