FAKULTÄT FÜR !NFORMATIK

Faculty of Informatics

Memetic Algorithms for Tree Decomposition

Diplomarbeit

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Software Engineering & Internet Computing

eingereicht von

Kevin Bader

Matrikelnummer 0726375

an der Fakultät für Informatik der Technischen Universität Wien

Betreuer: **Priv.-Doz. Dr. Nysret Musliu**

Wien, am 4.12.2014

(Verfasser) (Betreuer)

Technische Universität Wien, Karlsplatz 13, 1040 Wien, Austria. www.tuwien.ac.at

FAKULTÄT
FÜR !NFORMATIK

Faculty of Informatics

Memetic Algorithms for Tree Decomposition

Master's Thesis

submitted in partial fulfillment of the requirements for the degree of

Diplom-Ingenieur

in

Software Engineering & Internet Computing

by

Kevin Bader

Registration number 0726375

to the faculty of Informatics, Vienna University of Technology

Advisor: **Priv.-Doz. Dr. Nysret Musliu**

Vienna, Dec. 4th, 2014 ____________________ ____________________

(Author) (Advisor)

Erklärung zur Verfassung der Arbeit

Kevin Bader
Leystraße 163 Stiege 2 Tür 24, 1020 Wien

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, und dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe. Weiters habe ich alle Stellen der Arbeit, die anderen Werken im Wortlaut oder dem Sinn nach entnommen sind, unter Angabe der Quelle als Entlehnung kenntlich gemacht.

(Ort, Datum)	(Unterschrift Verfasser)

Danksagung

Ich möchte an erster Stelle meiner Mutter Gertrude dafür danken, dass sie mir mein Studium ermöglicht hat, und mir stets mit Rat und Unterstützung zur Seite gestanden ist. Ebenso möchte ich Judith danken, die mir während meiner ganzen Studienzeit immer wieder den Rücken gestärkt hat, und ohne die ich heute nicht dort wäre wo ich jetzt bin. Dank gilt auch Eva und Inge, deren finanzielle und teigwarenartige Unterstützung Gold wert waren im bescheidenen Studentendasein der letzten Jahre. Dankbar bin ich auch Pascal, Caro, Thomas und Jakob für ihre Zeit und Geduld.

Natürlich möchte ich mich auch bei meinem Betreuer, Dr. Nysret Musliu, für die umfassende und motivierende Betreuung bedanken. Danke auch an Toni Pisjak für den flinken technischen Support während der Durchführung der Experimente.

Zusammenfassung

Baumzerlegungen kleiner Breite stellen ein mächtiges Werkzeug dar, um der Komplexität schwieriger Probleme entgegenzutreten. Instanzen des Bedingungserfüllungsproblems können beispielsweise in polynomieller Zeit gelöst werden, wenn der zugrundeliegende Graph eine geringe Baumweite aufweist. Es gibt bereits mehrere exakte Algorithmen und Heuristiken, die gute obere Schranken für die natürliche Baumweite eines Graphen finden können. Da der Abstand zwischen unterer und oberer Schranke für viele Instanzen aber noch sehr groß ist, sind solche Algorithmen nachwievor Gegenstand aktueller Forschung. Diese Arbeit untersucht die Brauchbarkeit memetischer Algorithmen in Bezug auf dieses Problem.

Diese Arbeit präsentiert drei neue memetische Algorithmen für das Finden von Baumzerlegungen möglichst kleiner Breite. Die erste Variante verwendet eine zufällig gewählte Ringstruktur, um eine Population von Lösungen zu organisieren. Die Auswahl und Kombination der Individuen hängt von ihrer Umgebung in der Ringstruktur ab. Der zweite Algorithmus ist ein Hybrid zwischen einem genetischen Algorithmus, der mit einer Plus Strategie und Elitarismus arbeitet, und einer Heuristik, die auf lokaler Suche basiert, und in jeder Generation für das Verbessern der besten Lösungen verwendet wird. Die letzte Variante verwendet einen bestehenden genetischen Algorithmus, erweitert um lokale Suche, mit der in jeder Generation ein zufällig gewählter Teil der Population verbessert wird.

Alle drei Varianten wurden mit aktueller Parameter Tuning Software auf die Instanzen der Second DIMACS Implementation Challenge optimiert. Um den Einfluss der Parameter besser verstehen zu können, wurden die Korrelationen zwischen Parametereinstellungen und der Qualität der Ergebnisse untersucht und visualisiert.

Um die Leistung der Algorithmen abseits der Benchmark Instanzen bewerten zu können, wurden sie in einem umfassenden Experiment mit einem separaten Satz Instanzen getestet und verglichen. Die Ergebnisse wurden mit statistischen Tests auf ihre Signifikanz hin überprüft. Auch wenn unsere memetischen Algorithmen die aktuell besten Heuristiken nicht dominieren, erweisen sie sich für die meisten Instanzen durchaus als kompetitiv. Weiters konnten die bis dato besten oberen Schranken für 8 DIMACS Instanzen von den memetischen Algorithmen verbessert werden.

Abstract

A tree decomposition of small width represents a valuable tool for tackling hard problems. For example, instances of constraint satisfaction problems can be solved in polynomial time if their underlying graph can be transformed into a tree decomposition of small width. There exist several exact algorithms and heuristics for finding tree decompositions of small widths for a given graph. However, developing new methods for finding good upper bounds for tree decompositions is still an important research issue. This thesis investigates the application of memetic algorithms, which have, to the best of our knowledge, not yet been applied to tree decompositions.

We propose and implement three new memetic algorithms for finding tree decompositions of small width. The first algorithm organizes its population of solutions in a randomly initialized ring structure, which is used to employ selection and recombination among the individual solutions according to their vicinity. Each generation is collectively improved using iterated local search. The second algorithm is a hybrid between a genetic algorithm that uses elitism for selection, and iterated local search, which is used to improve the best individuals of every generation. Finally, the third algorithm implements an existing genetic algorithm design and also incorporates iterated local search, which is applied to a random fraction of each generation.

All three variants have been tuned using state-of-the-art parameter tuning software on instances of the Second DIMACS Implementation Challenge. In order to achieve a better understanding of the algorithms, the correlation between parameter settings and solution quality has been examined.

We offer an extensive comparison, which relates the performance of our memetic algorithms to state-of-the-art solvers for tree decomposition. Featuring a set of instances different to those used for parameter tuning, the comparison aims at providing real-world validity. The results have been confirmed using statistical significance tests. We show that our memetic algorithms do not dominate the state-of-the-art algorithms for this problem, but they prove to be competitive on most instances. Furthermore, one of our algorithms has been able to improve 8 best known upper bounds for the benchmark instances of the Second DIMACS Implementation Challenge.

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

Over the past century, computers have enabled us to quickly solve more and more problems that had previously been too intricate to be solved at all. With computing power increasing constantly, it is hard to believe that there are still problems left that still are hard to be solved. Many interesting problems, however, seem to be not only hard to solve: in fact, the majority of computer scientists thinks that many interesting problems will remain intractable forever.

The good news is that even for problems that are proven to be NP-hard (a class for computationally hard problems), often instances of these problems can be solved efficiently nevertheless. If the instance is small, it is possible to simply test all possible solutions in order to find the best one. For larger instances, often structural properties allow for an efficient search, regardless of the instance's size. In this work, we are concerned with exploiting one particular structural property called *treewidth*. The problem and the structural property are best explained using an example.

Consider a typical constraint satisfaction problem (CSP) that deals with the following scenario. Assume you are organizing a holiday camp for kids. Some kids know each other already, some don't. You plan to assign the kids to their dorms in such a way that the kids may become acquainted with other kids they do not yet know. Figure 1.1 illustrates the relationships between the kids.

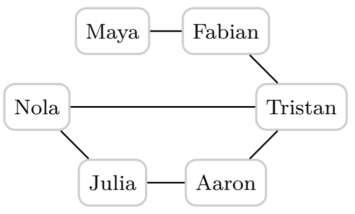


Figure 1.1: *Example CSP graph: kids and their relationships.*

The drawing shows the relationships as a graph, where the children are represented by vertices. The edges between them show the acquaintances, that is, if two kids know each other, there is an edge between them in the graph.

So our plan is to assign the kids to rooms such that no two kids in any room have an edge between them in the graph. In this example, is this possible with two rooms? Indeed, it is: we could give one room to Tristan, Julia and

Maya, and another to Nola, Aaron and Fabian.

This is a variant of a problem called Graph Coloring Problem, also known as Chromatic Number, which asks, given a graph, if it is possible to assign k colors to the vertices in the graph, such that each edge only connects vertices that have different colors. In our example, we have asked if this is possible with $k = 2$. With more than two colors, the decision version of the Graph Coloring Problem—e.g., is there a k-coloring with $k = 3$?—is already significantly harder to solve; in fact, for $k \geq 3$ it is known to be NP-complete (see [48] for an introduction to complexity theory). The optimization version, i.e., computing the chromatic number, is NP-hard as well. To give an idea what this means, consider the example again. Since we have 6 kids and two rooms, there are $2^6 = 64$ possible kids-to-rooms assignments. Despite these 64 possibilities, we were quick in finding a solution to the example here, so we would expect that it would not be that hard to solve the same problem for, say, a class of 25 pupils. However, when asking whether we can assign the 25 pupils to $k = 5$ rooms in the same way, we already face $5^{25} = 2.9802322 \cdot 10^{17}$ possible assignments. Assuming that evaluating a single assignment takes a computer one hundredths of a second, testing all possible assignments would still take 94502544 years. This demonstrates the complexity of NP-complete problems, where the runtime grows exponentially with the size of the problem instance.

Fortunately, we might still be able to solve such an instance in a reasonable amount of time, if it has a special structure, as we will see below.

Typically, the Graph Coloring Problem is formulated as a CSP:

Definition 1.1 (CSP)**.** A CSP is a triple $\langle V, D, C \rangle$, where $V = \{v_1, \ldots, v_n\}$ is a set of variables, $D = \{d_1, \ldots, d_n\}$ a set of finite domains (which contain the values that can be assigned to the variables in V, respectively), and $C = \{c_1, \ldots, c_m\}$ a set of constraints (each constraint is given as a pair of a set of k variables $\in V$ and a k-ary relation that represents the actual constraint on these variables). A solution to a CSP is an assignment of values to all variables, such that all constraint relations are fulfilled.

In the case of the example above, we could use one variable for each kid ($V = \{\text{Nola}, \text{Tristan}, \ldots\}$), and use the room numbers as domains for these variables (e.g., $\text{Nola} = 2$ means that Nola is assigned to the room 2). The constraints would be formulas that allow two variables to have the same number only if they are not connected with an edge in the underlying graph.

Tree Decomposition and Treewidth

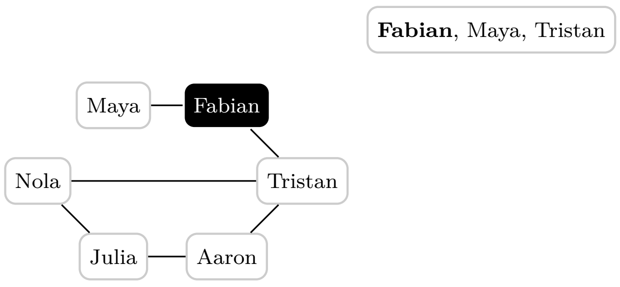
It turns out that a CSP instance is tractable if it features certain structural properties. Either it is tractable due to a restricted structure of the contraint scopes, or due to particular properties of the constraint relations [22, 49]. In order to identify and solve tractable CSPs instances, decomposition methods may be used. In this work, we focus on *tree decompositions*, and a property called *treewidth.* Creating a tree decomposition of a graph means to translate the original graph into another graph, the tree decomposition, according to a set of rules. The formal definitions of tree decomposition and treewidth are given in Section 2.1 on page 9.

Interestingly, a small treewidth causes shorter running times: if the underlying graph has bounded treewidth, the problem can be solved in $\mathcal{O}(n \cdot d^{t+1})$, where n is the number of variables, d is the maximal domain size of any variable, and t is the treewidth of the instance [52]. Clearly, the treewidth has a great impact on the runtime (as t is in the exponent); therefore the goal is to find a tree decomposition of minimal width, which can then be used to solve the original problem.

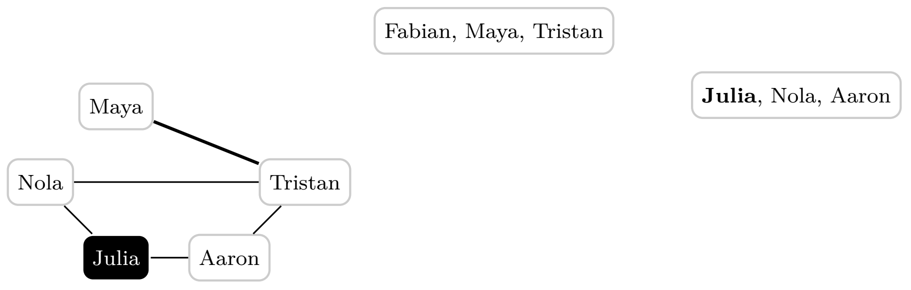
Relating to the school class problem: if we can find a tree decomposition of the underlying acquaintance graph with a sufficiently small width, we are able to solve it quickly.

In order to illustrate how tree decompositions may be derived, we consider the first example again, and develop one possible tree decomposition off the original graph. A common way to generate a tree decomposition is by *vertex elimination*, where vertices are removed from the original graph one by one, according to an *elimination ordering.* An elimination ordering then explicitly defines the resulting tree decomposition. To illustrate the elimination process, we use the following elimination ordering with the previous example: Fabian, Julia, Maya, Aaron, Nola, Tristan. Please note that this ordering is completely arbitrary. Most of the algorithms that solve this problem essentially do the same: they calculate the width of tree decompositions using the steps that are illustrated in the following example, according to (intelligently) guessed elimination orderings. The aim of the algorithms is to be good at finding these orderings, so that the width of the resulting tree decomposition is as small as possible.

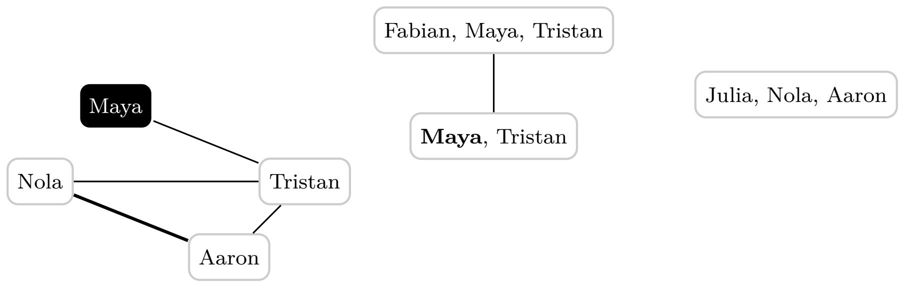
We start by eliminating the first vertex: Fabian. When removing a vertex from the original graph (on the left), we create a tree node in the tree decomposition (on the right) that contains the eliminated vertex and all its neighbors. When removing Fabian, we create a node that contains Fabian, Maya and Tristan:



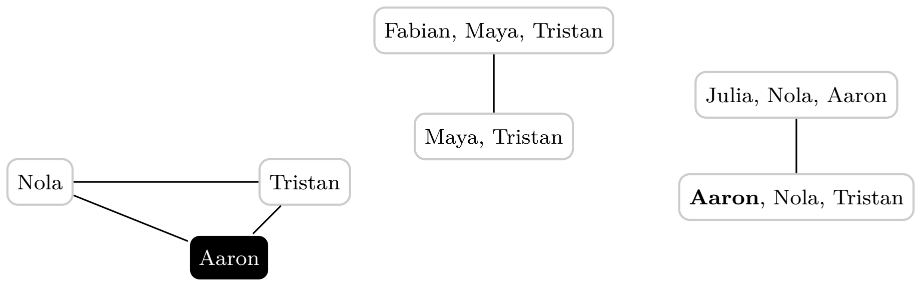
After removing a vertex, we connect in the original graph all its previous neighbors with each other. In the case of Fabian, we connect Maya and Tristan. Then we remove the next vertex: Julia.



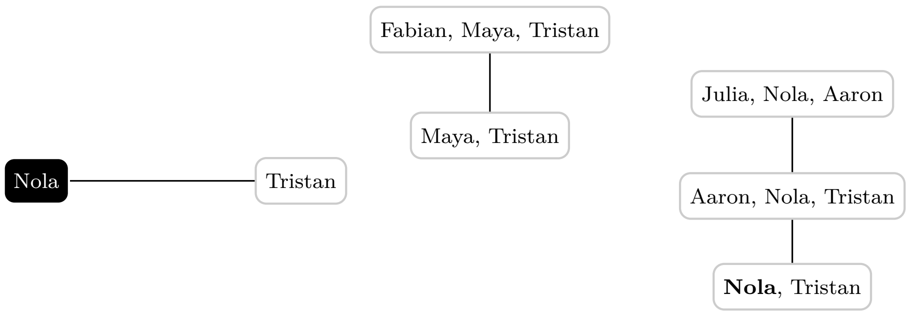
A node in the tree decomposition should be connected to the *next* node that is created because of the elimination of a vertex that occurs in it. In this case, Julia does not appear in the first node, so the two nodes are not connected. We continue with Maya.

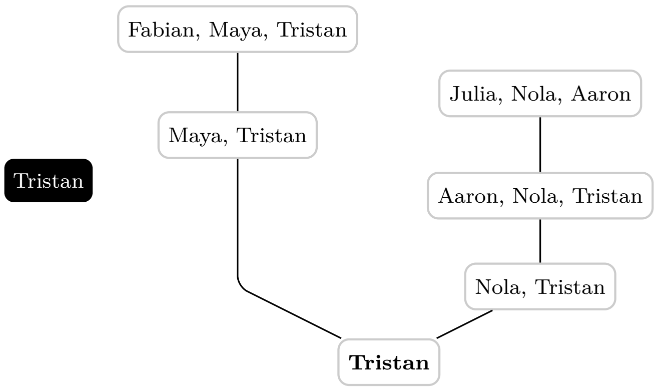


Since Maya is contained in the first node, and the first node has not been connected to a newer node yet, we connect the two nodes. Next: Aaron.



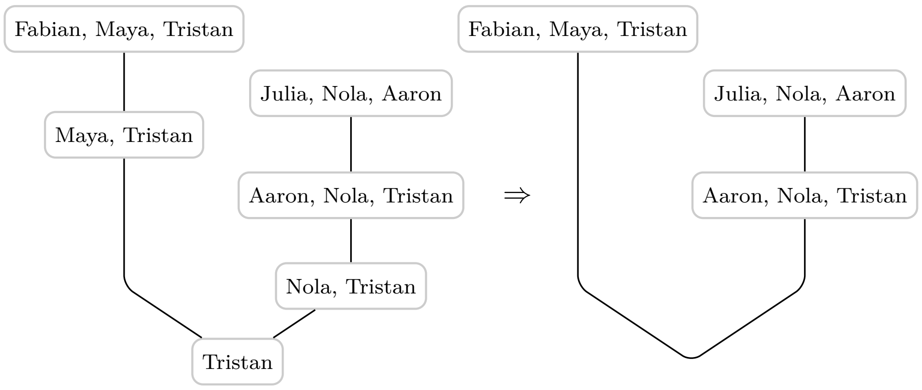
Aaron's tree node too gets connected to a previous node. The two remaining vertices are Nola and Tristan.





Tristan is present in two nodes, which both have not been connected to any newer nodes yet, so Tristan's node gets connected to both.

Finally, we may delete all nodes where the vertices are fully contained in a neighbor node:



The intuition behind this tree decomposition is that the nodes represent *subproblems*, and relating subproblems are connected to each other. In this example, we can observe that assigning a room to Tristan influences the possible solutions for the Aaron-Nola-Tristan node; similarly, the assignment for the latter affects the Julia-Nola-Aaron node. The idea is to divide the problem into smaller subproblems, and by concentrating on the subproblems, solve the original problem more efficiently.

Unfortunately, just as solving the original problem, finding the treewidth of a graph (i.e., the minimal width over all tree decompositions; see Definition 2.1) is NP-hard as well [5]. Consequently, the purpose of the optimization heuristics that are presented in this thesis is to find good upper bounds on the treewidth of given graphs.

1.1 Research Questions of This Work

There are different approaches for finding good upper bounds on the treewidths of graphs, as will be discussed in Chapter 2. To the best of our knowledge, however, memetic algorithms, which have been applied successfully to other problems in the past, have not yet been applied to this problem.

This work's objectives are:

- The development of new memetic algorithms for minimizing upper bounds on the treewidth of graphs.
- An evaluation of the developed memetic algorithms on benchmark instances from the literature and the comparison with state-of-the-art algorithms for tree decompositions.

1.2 Main Results

The main contributions of this work are:

- We propose three memetic algorithms for tree decomposition: MA1, MA2 and MA3. MA1 organizes a population of solutions in a randomly initialized ring structure, which is used for employing selection and recombination among the individual solutions according to their vicinity. Each generation is collectively improved using iterated local search. MA2 is a hybrid between a genetic algorithm that uses elitism for selection, and iterated local search, which is used to improve the best individuals of every generation. Finally, MA3 implements an existing genetic algorithm design and also incorporates iterated local search, which is applied to a random fraction of each generation. Their parameters have been tuned using state-of-the-art parameter tuning software, and the correlation between those parameters and the corresponding solution quality has been studied.
- Our algorithms have been compared with state-of-the-art solvers for tree decomposition. The memetic algorithms cannot outperform them, but they prove to be competitive as they perform equivalently on many benchmark instances. One of our algorithms (MA3) is able to improve 8 best known upper bounds for instances of the Second DIMACS Implementation Challenge [14].

- Public open-source implementations have been written from scratch and released under GPLv3. Code quality is ensured by providing high test coverage.

1.3 Further Organization

In the following, Chapter 2 discusses the related work. Then Chapter 3 introduces our memetic algorithms. Chapter 4 elaborates on parameter tuning, and also inspects possible parameter correlations. In Chapter 5 the experimental evaluation of the memetic algorithms is presented. Finally, Chapter 6 gives a conclusion.

Additionally, Appendix A list results from the algorithm validation that have been omitted in Section 5 for brevity.

Chapter 2

Related Work

In this Chapter we present literature related to tree decomposition.

2.1 Tree Decomposition and Treewidth

Treewidth has first been introduced by Robertson and Seymour in 1986.

Definition 2.1 (Tree Decomposition [51, 33])**.** Let $G = (V, E)$ be a graph. A *tree decomposition* of G is a pair $(T, \mathcal{X})$, where $T = (I, F)$ is a tree with node set I and edge set F, and $\mathcal{X} = \{X_i : i \in I\}$ is a family of subsets of V, one for each node of T, such that

(i) $\bigcup_{i \in I} X_i = V$

(ii) for every edge $vw \in E$, there is an $i \in I$ with $v \in X_i$ and $w \in X_i$, and

(iii) for all $i, j, k \in I$, if j is on the path from i to k in T, then $X_i \cap X_k \subseteq X_j$.

The *width* of a tree decomposition is $\max_{i \in I} |X_i| - 1$. The *treewidth* of a graph G, [...], is the minimum width over all possible tree decompositions of G.

Tree decompositions have successfully been used in solutions to many different problems, including partial CSPs [35], the Frequency Assignment Problem [34], Inference Problems in Probabilistic Networks [39], or the Vertex Cover Problem for planar graphs [2]. More recently, it has been used for finding differentially regulated subgraphs in biochemical networks [27], or the analysis of social graphs [1]. A survey on different techniques for computing treewidth, along with proofs of the underlying theorems, may be found in [10]. Recent work suggests that the treewidth might not be the most important feature of tree decompositions for solving CSPs, and related concepts are being introduced, e.g., bag-connected tree decompositions [29].

In the following, different approaches to computing minimal-width tree decompositions of graphs are discussed.

2.1.1 Approximation Algorithms

Polynomial-time approximation algorithms have been found as well. Instead of looking for an optimal solution, they guarantee that their solutions meet a certain level of quality. There are approximation algorithms that achieve a ratio of $\mathcal{O}(\log k)$, when k is the actual treewidth of the input graph; that is, they are able to find tree decompositions of a width smaller than $\mathcal{O}(k \log k)$ [3, 12].

There are also several exponential-time approximation algorithms that are able to find tree decompositions of width at most ck, where c is some constant; or they tell that the actual treewidth is higher than some given k. For example, see [3, 7, 37].

2.1.2 Polynomial-Time Greedy Heuristics

Polynomial-time greedy heuristics do not aim at finding the optimal solution, but rather look for reasonable solutions that can be found in a short amount of time. "Greedy" means that the heuristics build up an elimination ordering one vertex at a time, without ever revoking their decisions. The greedy heuristics for tree decompositions typically differ only in the criterion by which the next vertex is selected.

A simple representative is the Minimum Degree heuristic, which always selects the vertex that currently has the minimum number of unselected neighbors. The Minimum Fill-in heuristic (also known as the min-fill heuristic), on the other hand, selects the vertex that causes the minimal number of additional edges when being removed (recall from Chapter 1 that erasing a vertex causes additional edges that connects all of its former neighbors with each other). Another one is Maximum Cardinality Search (MCS) [58]. When choosing the next vertex for inclusion into the elimination ordering, MCS counts the number of neighbors (in the original graph) that are already in its current ordering; it then selects the vertex with the most neighbors already included in the ordering (it initializes the ordering with a randomly selected vertex). See [33] for further information and additional polynomial-time greedy heuristics.

2.1.3 Exact Algorithms

In [19], an exponential time algorithm that runs in $\mathcal{O}^*(1.9601^n)$ time is presented (for an explanation of this notation and a general introduction into exponential time algorithms, see [61]). Exponential time algorithms that run in $\mathcal{O}^*(2^n)$, can be found, for instance, in [26]. Branch-and-bound algorithms have independently been developed in [21] and [6]. They tend to be very efficient, especially for smaller instances. The algorithm in [21] is designed to be an anytime algorithm, which means that it will always return a valid solution, even when stopped before finding the optimum. Another branch-and-bound like approach is introduced in [53], based on A* search.

2.1.4 Metaheuristics

Algorithms that perform optimization can either be exact or heuristic. Exact algorithms will find an *optimal* solution in a finite amount of time. Heuristics, on the other hand, lack this guarantee; the solutions they return may in fact be anything between optimal and very poor. The motivation to use heuristics stems from complexity: many interesting problems—theoretical and real-world alike—have been shown to remain forever intractable to exact algorithms, regardless of the ever increasing computational power, simply due to unrealistically large running times. [54]

So a heuristic is an algorithm that produces a result to an optimization problem on a best-effort basis. A metaheuristic is then a template for constructing heuristics:

> "A metaheuristic is a high-level problem-independent algorithmic framework that provides a set of guidelines or strategies to develop heuristic optimization algorithms. The term is also used to refer to a problem-specific implementation of a heuristic optimization algorithm according to the guidelines expressed in such a framework."
> [54, 55]

The last sentence suggests why there is some confusion about the term: a metaheuristic is used as a template for designing a heuristic to solve a problem, but this heuristic is then often referred to as metaheuristic as well.

For a general overview of metaheuristics, see [55]. History and trends of metaheurstics are highlighted in [54]. Information on metaheurstic methods in the context of tree decompositions may be found in [25], which features genetic algorithms (GAs) [53, 46, 38], ant colony optimization [23, 24], and iterated local search (ILS) [45]. Other metaheuristic approaches include simulated annealing [32], and tabu search [16].

Iterated Local Search

ILS is a local-search metaheuristic [56, 55]. At its core, it tries to find a good solution by applying small changes (*moves*) to a solution so as to gradually improve it. Eventually, this *local search* ends up with a *local optimum*, which means that there is no better solution available that could be obtained using these small changes. In order to avoid getting stuck in such a local optimum, ILS from time to time *perturbates* the solution—it applies a large, random change. Due to the perturbation, the local search continues in a different part of the search space, again improving its solution gradually using small changes.

Genetic Algorithms

A GA [55] belongs to the family of *evolutionary algorithms*, which mimic evolutionary processes in biology. Evolutionary algorithms in turn are *population-based*, which means that instead of alternating a single solution, they consider multiple solutions (the *population*) in parallel. In a GA, new solutions are derived during search by selecting solutions (*individuals*) from the population and recombining them, using *crossover operators*. In addition, possibly *mutation* is applied as well, which is a small random change applied to a solution, using a *mutation operator*.

2.1.5 Preprocessing

In [9], Bodlaender et al. provide theoretical evidence of the importance of preprocessing rules for achieving best results. Preprocessing, they argue, has proven to be valuable, especially for real-world instances. As the focus of this work is the comparison between algorithms, preprocessing has not been applied. It remains an open question, however, if different algorithm benefit equally from preprocessing.

2.2 Memetic and Hybrid Algorithms

In [43], Moscato introduces the concept of *memetic algorithms*, based on the notion of a *meme* as described by Dawkins [18]. Moscato suggests that, in the realm of memetic algorithms, a meme may be seen as “a structure with internal consistency”; one or many such memes may comprise a solution. A memetic algorithm after Moscato is an evolutionary algorithm, and therefore uses a population of individuals, rather than a single current solution, to traverse the search space. On top of that, each individual applies a localsearch (LS) heuristic on its own solution. Eventually, the resulting local improvements are distributed among the population, according to mechanisms that are similar to GA’s crossover, mutation, and selection. The paper suggests to use alternating phases for each individual, as described in Section 3.1 on page 15.

Additional design considerations can be found in a more recent book chapter on memetic algorithms [44]. They suggest tuning the LS technique to the characteristics of the search space, and choosing the crossover and mutation operators accordingly. Apparently, inadequate parameter configuration can cause a easily solvable instance to turn non-polynomially solvable. Consequently, (self-)adaptive mechanisms that tune the algorithm to the instance at hand can be beneficial. Finally, the selection of individuals to run LS on can be an important choice as well, when not running LS on the whole population; for instance, by selecting the best individuals, or by using a stratified approach that selects one individual from each “quality level”.

Successful implementations of memetic algorithms have been reported for, e.g., the Quadratic Assignment Problem [41], the Travelling Salesman Problem and the Protein Folding Problem [36], the Generalized Travelling Salesman Problem [11], the Graph Coloring Problem [40, 20], and many others. Another real-world application can be found in [59, 60] for the break scheduling problem, which is concerned with finding a shift schedule that conforms to various constraints as closely as possible. They define a meme as interfering shifts in such a schedule, using the sum of constraint violations that the meme causes as its fitness.

The memetic algorithm (MA) for the Graph Coloring Problem by Galinier and Hao [20] can be described as a GA with random selection, full crossover, and complete local search instead of mutation. In other words, for every generation *all* individuals are created using a crossover operator, with the parents selected at random (as opposed to the usual k-tournament selection). After that, *all* individuals undergo local search *instead* of mutation. Their approach has been tested implicitly in this work, by allowing certain parameter ranges for MA2. In particular, the relevant parameters are

$$\textit{tournament size} = 1, \; P_{\text{crossover}} = 1, \text{ and } \textit{localsearch fraction} = 1.$$

Notably, the original implementation utilized a different crossover operator and local search procedure; substituting those with what is used during the other experiments, however, is done deliberately, as it ensures that the results reflect the quality of the particular approach, rather than the quality of the crossover operators and local search techniques.

A definition of, and a survey on, hybrid metaheuristics in general can be found in [55] and [8], respectively. A taxonomy for classifying hybrid algorithms can be found in [31].

Chapter 3

Memetic Algorithms for Treewidth Optimization

We have developed three memetic algorithms for finding tree decompositions of small treewidth. Section 3.1 presents MA1, which closely resembles the original design considerations of Moscato [43], along with its concept of partners and opponents. Using a randomly initialized ring structure, selection and recombination is applied with respect to the individual's neighborhood. Section 3.2 on page 23 introduces MA2, which is more similar to a traditional GA, features elitism for selection, and includes a local search heuristic to improve the best individuals of each generation. Finally, Section 3.3 on page 25 combines a GA design, borrowed from [53, 46], with iterated local search.

In order to ease comparison, all three GAs share the implementation of their ILS subroutine, which is based on [45]. For representing solutions they use elimination orderings, i.e., ordered lists of distinct vertex labels. The fitness of a solution is determined by applying vertex eliminiation to the given graph, similar to what is described in Chapter 1, while recording the size of the largest clique during the process. The implementation, which strives to be as efficient as possible, is based on the fitness functions used in, e.g., [45, 23, 24].

3.1 MA1

MA1 is a memetic algorithm that is similar to the algorithm for the Travelling Salesman Problem in [43]. It shares the concepts of population, individuals, and crossover with GAs, but it imposes a ring structure on the population (see below), as opposed to GAs, where the population is unstructured. MA1's pseudo code is displayed in Algorithms 3.4 to 3.6 on pages 20–22. The ILS heuristic is presented in Algorithms 3.1 on the following page, 3.2 on page 17, and 3.3 on page 18. Modifications to the design found in [45] include a different perturbation size that depends on the size of the population.

Algorithm 3.1 ILS, Part I

```
global $degradation\ tolerance \geq 0$, set to 3
global $maximal\ nonimproving\ outer\ steps > 0$
global $maximal\ nonimproving\ inner\ steps > 0$, set to 10
global $maximal\ perturbation\ repetitions > 0$, set to $0.05 * number\ of\ vertices$
```

▷ continued on page 17...

As the basis of memetic algorithms, Moscato utilizes Richard Dawkins' definition of a *meme* as "a unit of imitation in cultural transmission" [18]. The idea of using memes for algorithms is similar to the idea of gene transmission in GAs, but memes are meant to be small consistent parts of a solution rather than a solution as a whole; additionally, memes are meant to spread faster, involving multiple individuals, whereas in GAs recombination typically causes exactly two offsprings off of exactly two individuals.

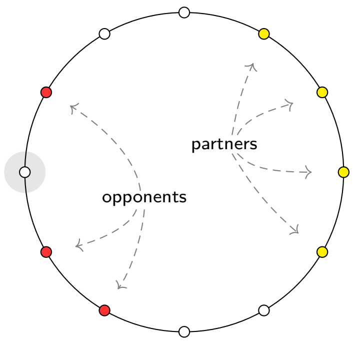


Figure 3.1: *Examplary ring topology for MA1, where the circles represent individuals. The colored circles depict the opponents and partners of the marked individual.*

The concept of memes does not seem to match the problem at hand very well (at least using elimination orderings as representation), since a part of an elimination ordering does not make sense on its own. Nevertheless, it is possible to apply the concept, such that parts of the orderings are coined memes and exchanged between individuals. In fact, since location and size of a part—the meme—are arbitrary, crossover operators can be applied in the usual way: during crossover, randomly selected ordering positions comprise a meme, and those memes are exchanged to yield new individuals.

MA1's main idea is the emphasis on communication between individuals, as a reminiscence to how memes spread between people in the real world[1]. Each individual communicates differently with three distinct groups of individuals: its partners, its opponents, and the rest of the population. While there is no communication with the latter group, partners are used for *cooperation*, and opponents for *competition*.

Cooperation is what recombination is to a GA. In fact, the implementation of MA1 uses the same position-based crossover (POS-crossover) operator for cooperation as MA2 (Section 3.2) and MA3 (Section 3.3) do for recombination.

[1]In a multi-threaded architecture, this communication could happen in parallel, like in the real world, which could potentially lead to great speedups in performance. For the sake of comparability to the other algorithms, however, all experiments have been executed using a single-threaded implementation.

Algorithm 3.2 ILS, Part II

▷ ...continued from page 16

```
function ILS
    solution ← random permutation of vertices
    best ← solution
    perturbation repetitions ← 1
    last candidate ← will hold the last iteration's candidate solution
    nonimproving steps ← 0
    while nonimproving steps < maximal nonimproving outer steps do
        candidate ← MIN-CONFLICTS TREE-SEARCH(solution)
        if fitness(candidate) = fitness(last candidate) then
            same solution counter ← same solution counter + 1
        end if
        last candidate ← candidate
        if fitness(candidate) < fitness(best) then              ▷ improvement!
            solution ← candidate
            best ← candidate
            nonimproving steps ← 0
            same solution counter ← 0
            perturbation repetitions ← 1
        else
            nonimproving steps ← nonimproving steps + 1
            if fitness(candidate) = fitness(best) then           ▷ quality unchanged
                solution ← candidate
                best ← candidate
            else if fitness(candidate) < fitness(best) + deg. tolerance then
                solution ← candidate             ▷ reuse the slightly worse solution
                                                              in the next iteration
            else
                solution ← best                  ▷ forget about the candidate and
                                                           go back to a better state
            end if
        end if
        if nonimproving steps > 0 and $\frac{\text{same solution counter}}{\text{nonimproving steps}} > 20\,\%$ then
            perturbation repetitions ← max(perturbation repetitions + 1,
                                            maximal perturbation repetitions)
        end if
        solution ← RANDOM MOVEMENTS(solution, perturbation repetitions)
    end while
    return best
end function
```

▷ continued on page 18...

Algorithm 3.3 ILS, Part III

▷ ...continued from page 17

```
function MIN-CONFLICTS TREE-SEARCH(start solution)
                                                    ▷ called “LS1” in [45]
    best ← start solution
    nonimproving steps ← 0
    while nonimproving steps < maximal nonimproving inner steps do
        candidate ← in best, swap the vertex that causes the largest clique
                    when eliminated during the triangulation (with ties broken
                    randomly) with another, randomly selected vertex
        if fitness(candidate) < fitness(best) then          ▷ improvement!
            best ← candidate
            nonimproving steps ← 0
        else
            nonimproving steps ← nonimproving steps + 1
        end if
    end while
    return best
end function

function RANDOM MOVEMENTS(solution, repetitions)
    for all $i \in \{1, 2, \ldots, repetitions\}$ do
        position 1 ← random position in solution
        position 2 ← random position in solution that is ≠ position 1
        solution ← move vertex in solution from position 1 to position 2
    end for
    return solution
end function
```

When an individual *proposes* to one of its partners, it subsequently replaces its partner with their child[2], that is, their crossover result.

Competetion is what selection is to a GA. When an individual sends a *challenge* to one of its opponents, the fitness of the two is compared. If the opponent's fitness is worse, it is replaced by a copy of the challenging individual; otherwise, the challenge is deemed fruitless and nothing happens.

The core of the algorithm is comprised of the following four steps, which are executed by each individual in a loop:

1. Improve own solution by applying local search
2. Compete with a random opponent
3. Improve own solution by applying local search
4. Cooperate with a random partner

[2]No pun intended.

Just as with people in the real world, who (should) take their time to think about what they have heard prior to spreading the news, the individuals in MA1 are allowed to improve themselves in-between interactions.

In order to select the partners and opponents for each individual, an artificial ring-like topology is imposed on the population, as shown in Figure 3.1 on page 16. Using this ring topology, each individual's opponents are its neighbors, and its partners are a subset of the population that is on the other side of the ring. Both opponents and partners are connected in this topology. Furthermore, opponents are selected symmetrically around the respective individual; accordingly, partners are selected symmetrically around the individual that is found opposite in the ring.

Moscato suggests that in each phase each individual issues exactly one request, and receives exactly one request. In MA1, this is implemented by choosing partners and opponents by *sampling without replacement.*

3.1.1 Positional Crossover Operator

MA1 makes use of a *crossover operator*, which is capable of creating a new solution based on two existing solutions. Crossover operators are problem specific. In [53, 46], various operators have been tested, and POS-crossover [57] has found to be the most effective one. We have chosen to use only this operator, as the main goal for this work is to assess the value of the combination of the involved (meta)heuristics, rather than the impact of different crossover operators.

A POS-crossover C may be generated from two solutions A and B using a two-step procedure. First, a randomly selected subset of A is copied to C, maintaining the positions. Then the remaining positions of C are filled up with the missing vertices, in the order in which they occur in B. For instances that contain more than 500 vertices, this second step is done using a hash-map containing the vertex positions, which is populated as a preprocessing step. This has been evaluated empirically.

The pseudo code of POS-crossover is shown as part of MA2 in Algorithm 3.8 on page 25.

Algorithm 3.4 MA1, Part I

```
global population size > 0
global partner fraction ∈ [0, 1]
global opponent fraction ∈ [0, 1]
global maximal nonimproving outer steps > 0                    ▷ for ILS
global maximal nonimproving inner steps > 0                    ▷ for ILS

function MA1
    generation ← population size random solutions
    for each individual in generation do                ▷ organize population
                                                          in a ring topology
        opponents(individual) ← the population size * opponent fraction nearest
                                (w.r.t. the ring topology) individuals
        partners(individual) ← the population size * partner fraction most distant
                               (w.r.t. the ring topology) individuals
    end for each
    best solution ← random individual in generation
    loop for 1000 seconds:
        for each individual in generation do
            individual ← ILS(individual)              ▷ Algorithm 3.1 on page 16
        end for each
        best solution ← generation's best solution if better than best solution
        generation ← COMPETITIONS(generation)
        for each individual in generation do
            individual ← ILS(individual)              ▷ Algorithm 3.1 on page 16
        end for each
        best solution ← generation's best solution if better than best solution
        generation ← COOPERATIONS(generation)
    end loop
    return best solution
end function
```

▷ continued on page 21...

Algorithm 3.5 MA1, Part II

▷ ...continued from page 20

```
function COMPETITIONS($generation$)
    for each $individual$ in $generation$ do
        $opponent$ ← CHOOSE OPPONENT($individual$)
        mark $individual$ as challenging $opponent$
    end for each
    for each $individual$ in $generation$ do
        resolve issued challenge
        if successful then
            store $individual$ with the challenged individual,
                        but do not replace just yet
        end if
    end for each
    for each $individual$ in $generation$ do
        if $individual$ has lost a successful challenge in the previous step then
            replace $individual$ with the ordering stored with it in the previous step
        end if
    end for each
    return $generation$
end function

function CHOOSE OPPONENT($individual$)
    loop
        $opponent$ ← randomly choose from opponents($individual$)
        if $opponent$ has not yet been challenged in this generation,
                    and does not challenge $individual$ in this generation then
            return $opponent$
        end if
    end loop
end function
```

▷ continued on page 22...

Algorithm 3.6 MA1, Part III

▷ ...continued from page 21

```
function COOPERATIONS(generation)
    for each individual in generation do
        partner ← CHOOSE PARTNER(individual)
        mark individual as proposing to partner
    end for each
    for each individual in generation do
        recombine the two individuals using crossover (POS-crossover)
        store with addressee of the proposal (do not replace just yet)
    end for each
    for each individual in generation do
        if individual has been proposed to successfully in the previous step then
            replace individual with the ordering stored with it in the previous step
        end if
    end for each
    return generation
end function

function CHOOSE PARTNER(individual)
    loop
        partner ← randomly choose from partners(individual)
        if partner has not yet received a proposal in this generation,
                    and does not propose to individual in this generation then
            return partner
        end if
    end loop
end function
```

3.2 MA2

Our second memetic algorithm is MA2, which is shown by Algorithms 3.7 on the next page and 3.8 on page 25 in pseudo code notation. MA2 can be described as a GA that uses the plus strategy for selection (i.e., it is impossible for both parents and their offspring to survive at the same time; see line 26), applying either crossover replacement (line 26) or mutation (line 28), but never both, to each individual in every generation. Additionally, iterated local search is applied to the generation's fittest individuals (lines 31–34), using the same implementation as in MA1.

MA2 uses *elitism*, meaning that the best individual in each generation is guaranteed to survive unalteredly, at least until the next selection phase. The other candidates for the following generation are selected using k-tournament (lines 21–22).

The parameters of MA2 are listed at the top of Algorithm 3.7. If the tournament size k is set to 1, the k-tournament selection is effectively transformed into random selection. When combining this random selection with a crossover probability of 1 and local search for each individual, the algorithm behaves very similar to the memetic algorithm that Galinier and Hao presented for the graph coloring problem [20]. Because of this similarity, their approach is not modeled explicitly; instead, it is assumed that the tuner will simply shift MA2's parameters to these extremes in the case that they are indeed profitable (as we will see in Chapter 4, this is not the case).

MA2 also uses POS-crossover as its crossover operator, as described in Section 3.1.1 on page 19.

A similar algorithm has previously been used for the break scheduling problem [59, 60].

3.2.1 Insertion Mutation Operator

MA2 uses a *mutation operator*, which applies a small modification to a given solution. A mutation operator is typically used in GAs for escaping local optima; the same holds for MAs. Again, we have chosen to use the mutation operator that has proven to be the most effective one in [53, 46].

The insertion mutation (IM) [42] that is used in two of the MAs simply moves one vertex from its position in an elimination ordering to another, randomly selected position.

The pseudo code of IM is shown in Algorithm 3.8 on page 25.

3.2.2 k-Tournament Selection

MA2 uses *k-tournament*, which is a common, well-performing [47] selection operator, for selecting the individuals that are carried from one generation to

Algorithm 3.7 MA2, Part I

```
global *population size* $> 0$
global tournament size $k > 0$                                   ▷ selection pressure
global $P_{\text{crossover}} \in [0, 1]$                 ▷ probability for an individual to be replaced
                                                  by crossover with another individual
global *localsearch fraction* $\in [0, 1]$
global *maximal nonimproving outer steps* $> 0$                          ▷ for ILS
global *maximal nonimproving inner steps* $> 0$                          ▷ for ILS

function MA2
    generation ← population size random solutions
    best solution ← random individual in generation
    loop for 1000 seconds:
        generation ← ADVANCE POPULATION(generation)
        elitist ← the generation's best individual
        if fitness(elitist) $\leq$ fitness(best solution) then
            best solution ← elitist
        end if
    end loop
    return best solution
end function

function ADVANCE POPULATION(generation)
    elitist ← the generation's best individual
    candidates ← {elitist} $\cup$
                    SELECT INDIVIDUALS(generation, population size $- 1$)
    for each candidate in candidates \ elitist do
        if random float $\in [0, 1) < P_{\text{crossover}}$ then
            partner ← some other, randomly selected individual $\in$ candidates
            candidate ← CROSSOVER(candidate, partner)
        else
            candidate ← MUTATE(candidate)
        end if
    end for each
    fittest ← the (population size $*$ localsearch fraction) fittest
                individuals $\in$ candidates
    for each individual in fittest do
        individual ← ILS(individual)                    ▷ Algorithm 3.1 on page 16
    end for each
    return candidates
end function
```

▷ continued on page 25...

Algorithm 3.8 MA2, Part II

```
                                                  ▷ ...continued from page 24
function SELECT INDIVIDUALS(generation, count)
    selection ← {}
    loop for count times:
                                                  ▷ k-tournament
        contenders ← k randomly selected, distinct individuals ∈ generation
        winner ← the contender with the smallest width
        selection ← selection ∪ {winner}
    end loop
    return selection
end function

function CROSSOVER(parent1, parent2)
                                                  ▷ POS-crossover
    selected positions ← randomly select a set of positions of the ordering
    child ← new elimination ordering where selected positions are copied from
            parent1
    child ← child with empty positions filled with the remaining vertices, in the
            order of their occurence in parent2
    return child
end function

function MUTATE(individual)
    individual ← randomly choose a position in the individual's elimination
                 ordering and move the corresponding vertex to another,
                 randomly selected position
    return individual
end function
```

the next. k-tournament randomly chooses *k* individuals from a generation and returns the individual with the highest fitness.

The pseudo code of k-tournament selection is also shown in Algorithm 3.8.

3.3 MA3

The third memetic algorithm is called MA3. MA3 is also a combination between GA and ILS, and has been implemented based on two successful designs, namely Genetic Algorithm for treewidth (GA-tw) [53, 46] for the GA part and the Iterative Heuristic Algorithm for treewidth (IHA) [45] related ILS algorithm that is also used by MA1 and MA2.

The algorithm, which is displayed in Algorithms 3.9 on the next page and 3.10 on page 27, uses ILS to improve a fraction of each generation. The members of this fraction are selected at random in every iteration.

MA3 uses the POS-crossover operator as described in Section 3.1.1, the IM operator as described in Section 3.2.1, and the k-tournament selection operator as described in Section 3.2.2.

Algorithm 3.9 MA3, Part I

```
global population size ← 2000
global localsearch fraction ∈ [0, 1]

function MA3
    generation ← population size random solutions
    best ← random individual in generation
    loop for 2000 generations:
        generation ← SELECT(generation)
        generation ← RECOMBINE(generation)
        generation ← MUTATE(generation)
        selection ← (population size ∗ localsearch fraction) individuals ∈ generation,
                    selected at random
        for each individual in selection do
            individual ← ILS(individual)                ▷ Algorithm 3.1 on page 16
        end for each
        elitist ← the generation's best individual
        if fitness(elitist) < fitness(best) then
            best ← elitist
        end if
    end loop
    return best solution
end function

function SELECT(generation)
    selection ← {}
    loop for population size times:
                                                        ▷ k-tournament with k = 3
        contenders ← 3 randomly selected, distinct individuals ∈ generation
        winner ← the contender with the smallest width
        selection ← selection ∪ {winner}
    end loop
    return selection
end function
```

▷ continued on page 27...

3.4 Implementation

The memetic algorithms have been implemented from scratch, using C++. To ensure code quality and achieve correctness, state-of-the-art code testing practices have been employed, using Goole Mock[3] and Google Test[4] as the

[3]https://code.google.com/p/googlemock/ [4]https://code.google.com/p/googletest/

Algorithm 3.10 MA3, Part II

▷ …continued from page 26

```
function RECOMBINE(generation)
                                        ▷ the crossover rate is set to 100 %
    for each pair of neighbors in generation do
        first parent ← first elimination ordering in pair
        second parent ← second elimination ordering in pair
        first child ← CROSSOVER(first parent, second parent)
        second child ← CROSSOVER(second parent, first parent)
        in generation, replace first parent with first child
        in generation, replace second parent with second child
    end for each
    return generation
end function

function CROSSOVER(parent1, parent2)
                                        ▷ POS-crossover
    selected positions ← randomly select a set of positions of the ordering
    child ← new elimination ordering where selected positions are copied from
            parent1
    child ← child with empty positions filled with the remaining vertices, in the
            order of their occurence in parent2
    return child
end function

function MUTATE(generation)
                                        ▷ the mutation rate is set to 30 %
    selection ← 30 % of generation, selected at random
    for each individual in selection do
        individual ← randomly choose a position in the individual's elimination
                     ordering and move the corresponding vertex to another,
                     randomly selected position
        in generation, replace original individual with mutated individual
    end for each
    return generation
end function
```

supporting toolkits. All other used libraries are taken from the Boost C++ Libraries[5] project. The source code is made available online, licensed under GPLv3. See Figure 3.1 for information needed to reproduce the results found in this work. Further details can be found in the repository's README file.

[5] `http://www.boost.org/`

Table 3.1: *Information for reproducing the results.*

Git repository	`https://github.com/kevinbader/MAtw`
Platform	`x86_64` GNU/Linux (tested on `Intel64`)
Compiler	g++ (GCC) 4.9.1 20140903 (prerelease)
C++ standard library	libstdc++ that comes bundled with GCC
Boost	1.56.0
CMake	3.0.2

Chapter 4

Parameter Tuning

Not unlike other metaheuristics, our algorithms exhibit many parameters, meant for tuning them to specific problems. To allow for a fair comparison between the metaheuristics, these parameters ought to be *tuned* first. In general, tuning a metaheuristic algorithm means to run it with different parameter settings, so as to find the best setting for a specific problem. A simple way to do that is by exhaustive search, which means invoking the algorithm with all possible combinations of all possible parameter values. Clearly, this does not scale; in the light of multiple parameters with large or even infinite sets of possible values, other methods seem more attractive. For this work, we facilitate an automatic parameter tuner called Sequential Model-based Algorithm Configuration (SMAC), which uses regression models to describe the dependence of a target algorithm's performance on its parameter settings [28].Given a set of instances for training, SMAC returns the best-performing parameter configuration it can heuristically find. Somewhat arbitrarily, a runtime limit of 1000 seconds has been imposed on each run during the tuning process, which represents a trade-off between the solution quality of an individual run and the solution quality of the parameter tuning (assuming that the overall runtime should not exceed a few weeks).

The training instances that have been considered in this work are 79 Graph Coloring instances, most of which stem from the Second DIMACS Implementation Challenge[1] [14]. The instances may be obtained, for instance, from [15].

Tables 4.1 to 4.3 on pages 30–30 show the instances of the training set. They list all instances of the respective category, along with the instance's number of vertices (|V|) and edges (|E|). Additionally, the number of samples gathered during parameter tuning is given, with the instance that has been involved in the most evaluations given in bold typeface.

[1]The Second DIMACS Implementation Challenges been concerned with the NP hard problems Maximum Clique, Graph Coloring, and Satisfiability, and has been organized between 1992 and 1993.

Table 4.1: *Random graphs used in his paper with Aragon, McGeoch, and Schevon, "Optimization by Simulated Annealing: An Experimental Evaluation; Part II, Graph Coloring and Number Partitioning", Operations Research, 31, 378–406 (1991). DSJC are standard (n, p) random graphs. DSJR are geometric graphs, and DSJR*.c complements of geometric graphs.*

			Number of Samples		
Instance Name	**|V|**	**|E|**	**MA1**	**MA2**	**MA3**
DSJC125.1	125	1472	17	24	19
DSJC125.5	125	7782	18	22	19
DSJC125.9	125	13 922	16	23	29
DSJC250.1	250	6436	21	18	17
DSJC250.5	250	31 336	25	30	22
DSJC250.9	250	55 794	30	24	29
DSJC500.1	500	24 916	14	16	16
DSJC500.5	500	125 248	24	19	25
DSJC500.9	500	224 874	30	**31**	28
DSJC1000.1	1000	99 258	17	17	19
DSJC1000.5	1000	499 652	23	25	15
DSJC1000.9	1000	898 898	**31**	34	26
DSJR500.1	500	7110	20	22	14
DSJR500.1c	500	242 550	29	32	**39**
DSJR500.5	500	117 724	21	15	18

Table 4.2: *Quasi-random coloring problem.*

			Number of Samples		
Instance Name	**|V|**	**|E|**	**MA1**	**MA2**	**MA3**
flat300_20_0	300	21 375	21	26	26
flat300_26_0	300	21 633	**30**	21	**32**
flat300_28_0	300	21 695	16	16	20
flat1000_50_0	1000	245 000	21	19	21
flat1000_60_0	1000	245 830	24	19	22
flat1000_76_0	1000	246 708	27	**32**	22

Table 4.3: *Latin square problem.*

			Number of Samples		
Instance Name	**|V|**	**|E|**	**MA1**	**MA2**	**MA3**
latin_square_10	900	307 350	**25**	**20**	**26**

Table 4.4: *Problem based on register allocation for variables in real codes.*

			Number of Samples		
Instance Name	**|V|**	**|E|**	**MA1**	**MA2**	**MA3**
`fpsol2.i.1`	496	11 654	18	23	18
`fpsol2.i.2`	451	8691	**24**	21	18
`fpsol2.i.3`	425	8688	21	**24**	**27**
`inithx.i.1`	864	18 707	17	20	18
`inithx.i.2`	645	13 979	15	20	21
`inithx.i.3`	621	13 969	**25**	**30**	**25**
`mulsol.i.1`	197	3925	24	24	**33**
`mulsol.i.2`	188	3885	**33**	**33**	22
`mulsol.i.3`	184	3916	22	20	19
`mulsol.i.4`	185	3946	20	19	24
`mulsol.i.5`	186	3973	22	18	20
`zeroin.i.1`	211	4100	20	20	21
`zeroin.i.2`	211	3541	23	19	16
`zeroin.i.3`	206	3540	**35**	**27**	**31**

Table 4.5: *Leighton graphs with guaranteed coloring size. A reference is F.T. Leighton, Journal of Research of the National Bureau of Standards, 84: 489–505 (1979).*

			Number of Samples		
Instance Name	**|V|**	**|E|**	**MA1**	**MA2**	**MA3**
`le450_5a`	450	5714	23	20	29
`le450_5b`	450	5734	23	24	19
`le450_5c`	450	9803	28	**31**	30
`le450_5d`	450	9757	21	18	23
`le450_15a`	450	8168	17	21	18
`le450_15b`	450	8169	17	21	20
`le450_15c`	450	16 680	**38**	29	**34**
`le450_15d`	450	16 750	13	24	24
`le450_25a`	450	8260	16	14	22
`le450_25b`	450	8263	19	24	22
`le450_25c`	450	17 343	23	25	20
`le450_25d`	450	17 425	16	19	13

Table 4.6: *Book Graphs from Donald Knuth's Stanford GraphBase. Given a work of literature, a graph is created where each node represents a character. Two nodes are connected by an edge if the corresponding characters encounter each other in the book. Knuth creates the graphs for five classic works: Tolstoy's Anna Karenina (anna), Dicken's David Copperfield (david), Homer's Iliad (homer), Twain's Huckleberry Finn (huck), and Hugo's Les Misérables (jean).*

			Number of Samples		
Instance Name	**|V|**	**|E|**	**MA1**	**MA2**	**MA3**
`anna`	138	986	**31**	25	15
`david`	87	812	25	17	17
`homer`	561	3258	18	31	23
`huck`	74	602	20	27	32
`jean`	80	508	30	**35**	**37**

Table 4.7: *Game Graph from Donald Knuth's Stanford GraphBase. A graph representing the games played in a college football season can be represented by a graph where the nodes represent each college team. Two teams are connected by an edge if they played each other during the season. Knuth gives the graph for the 1990 college football season.*

			Number of Samples		
Instance Name	**|V|**	**|E|**	**MA1**	**MA2**	**MA3**
`games120`	120	1276	**23**	**22**	**21**

Table 4.8: *Miles Graphs from Donald Knuth's Stanford GraphBase. These graphs are similar to geometric graphs in that nodes are placed in space with two nodes connected if they are close enough. These graphs, however, are not random. The nodes represent a set of United States cities and the distance between them is given by by road mileage from 1947. These graphs are also due to Kuth.*

			Number of Samples		
Instance Name	**|V|**	**|E|**	**MA1**	**MA2**	**MA3**
`miles250`	128	774	15	18	19
`miles500`	128	2340	**26**	23	**23**
`miles750`	128	4226	21	23	22
`miles1000`	128	6432	26	**28**	21
`miles1500`	128	10 396	22	23	23

Table 4.9: *Queen Graphs from Donald Knuth's Stanford GraphBase. Given an n by n chessboard, a queen graph is a graph on n^2 nodes, each corresponding to a square of the board. Two nodes are connected by an edge if the corresponding squares are in the same row, column, or diagonal. Unlike some of the other graphs, the coloring problem on this graph has a natural interpretation: Given such a chessboard, is it possible to place n sets of n queens on the board so that no two queens of the same set are in the same row, column, or diagonal? The answer is yes if and only if the graph has coloring number n. Martin Gardner states without proof that this is the case if and only if n is not divisible by either 2 or 3. In all cases, the maximum clique in the graph is no more than n, and the coloring value is no less than n.*

			Number of Samples		
Instance Name	**|V|**	**|E|**	**MA1**	**MA2**	**MA3**
queen5_5	25	320	16	17	15
queen6_6	36	580	26	23	21
queen7_7	49	952	40	26	28
queen8_12	96	2736	17	19	28
queen8_8	64	1456	23	19	21
queen9_9	81	2112	18	21	18
queen10_10	100	2940	**41**	28	**32**
queen11_11	121	3960	29	18	24
queen12_12	144	5192	28	**31**	25
queen13_13	169	6656	17	21	19
queen14_14	196	8372	27	21	28
queen15_15	225	10 360	24	23	26
queen16_16	256	12 640	18	25	20

Table 4.10: *Graphs based on the Mycielski transformation. These graphs are difficult to solve because they are triangle free (clique number 2) but the coloring number increases in problem size.*

			Number of Samples		
Instance Name	**|V|**	**|E|**	**MA1**	**MA2**	**MA3**
myciel3	11	20	20	18	22
myciel4	23	71	16	23	18
myciel5	47	236	23	17	26
myciel6	95	755	26	**32**	**30**
myciel7	191	2360	**35**	28	23

Table 4.11: *Class scheduling graphs, with and without study halls.*

			Number of Samples		
Instance Name	**|V|**	**|E|**	**MA1**	**MA2**	**MA3**
school1	385	19 095	**20**	17	**23**
school1_nsh	352	14 612	17	**20**	21

4.1 Tuning Results

This Section presents the results of parameter tuning. SMAC has applied our algorithms to the training set for 21 days[2] each.

4.1.1 MA1

Table 4.12: *MA1's parameters after parameter tuning by SMAC.*

Parameter	**Range**	**Result**
Population size (GA)	$[10, 2000] \subset \mathbb{N}$	1941
Partner population-fraction	$[0, 1] \subset \mathbb{R}$	.9171
Opponent population-fraction	$[0, 1] \subset \mathbb{R}$	.0072
Max. nonimproving outer steps (ILS)	$[1, 200] \subset \mathbb{N}$	193
Max. nonimproving inner steps (ILS)	$[1, 100] \subset \mathbb{N}$	84

Table 4.12 shows the tuning result for MA1. Note that there has been no restriction on these values that would prevent an individual to use all other individuals for both cooperation and competition, such that the two groups would overlap. Indeed, using the result of the parameter tuning, this is not the case. The opponent population-fraction seems to have turned out quite small, but considering the population size, each individual faces approximately 14 opponents each round. They are not so few indeed, especially when comparing this with the size of the k-tournament operator that is used for a similar purpose in GAs. Other than that, the tuning result does not seem overly surprising.

4.1.2 MA2

Table 4.13: *MA2's parameters after parameter tuning by SMAC.*

Parameter	**Range**	**Result**
Population size (GA)	$[10, 2000] \subset \mathbb{N}$	15
Tournament size k	$[1, 5] \subset \mathbb{N}$	3
$P_{\text{crossover}}$	$[0, 1] \subset \mathbb{R}$	.9162
Localsearch fraction	$[0, 1] \subset \mathbb{R}$	.4020
Max. nonimproving outer steps (ILS)	$[1, 200] \subset \mathbb{N}$	82
Max. nonimproving inner steps (ILS)	$[1, 100] \subset \mathbb{N}$	94

[2]Arbitrarily chosen as a seemingly *large-enough*, and thus reasonable, value.

Table 4.13 on the preceding page displays the tuning result for MA2. The tuner has chosen a configuration that causes intense localsearch at the cost of a smaller population size—a trade-off that goes with limiting MA2's runtime.

4.1.3 MA3

Table 4.14: *MA3's parameters, with the ranges used by SMAC, and its output.*

Parameter	**Range**	**Result**
Population size	$[10, 2000] \subset \mathbb{N}$	92
Localsearch fraction	$[0, 1] \subset \mathbb{R}$	.7135
Max. nonimproving outer steps (ILS)	$[1, 200] \subset \mathbb{N}$	4

In order to reduce the number of parameters for tuning, the original configuration of GA-tw and IHA are used for the respective, similar parts of MA3. Consequently, there are three parameters relevant to the hybrid: the size of the population, the number of iterations ILS should run when invoked, and the fraction of the population ILS is applied to. Table 4.14 shows the corresponding ranges that have been used during parameter tuning, along with the tuning result. With relatively few individuals and a high localsearch rate, the found configuration favors the ILS part of the algorithm over its GA part.

4.2 Parameter Correlation

For each variant, the correlation between its parameter settings and the corresponding result is explored, using the data gathered during parameter tuning. Instead of applying statistical tests we utilize graphical representation, for the small samples sizes prevent us from accounting for confounding variables properly. Handling these confounders would be important, however, since the involved parameters can arguably be expected not to be independent from one another. Consequently, the correlation results can only give an overview of the relationships, and should not be considered to be exact.

The correlation diagrams include scatter plots of the samples that have been gathered during the tuning process; additionally, locally weighted scatterplot smoothing (LOWESS) curves [17] are meant to show a (nonlinear) trend line. Instance selection has been done to avoid showing 79 similar plots. Instead, so as to maximize space efficiency without losing information, those instances have been selected that have been used by the tuner the most, one for each instance type (identified by the name of the instance).

4.2.1 MA1

For MA1, the plots in Figures 4.1 to 4.14 on pages 36–42 do not show great disposedness towards any particular parameter. Nevertheless, they suggest that with this variant, smaller population sizes and higher localsearch intensities are beneficial to the outcome.

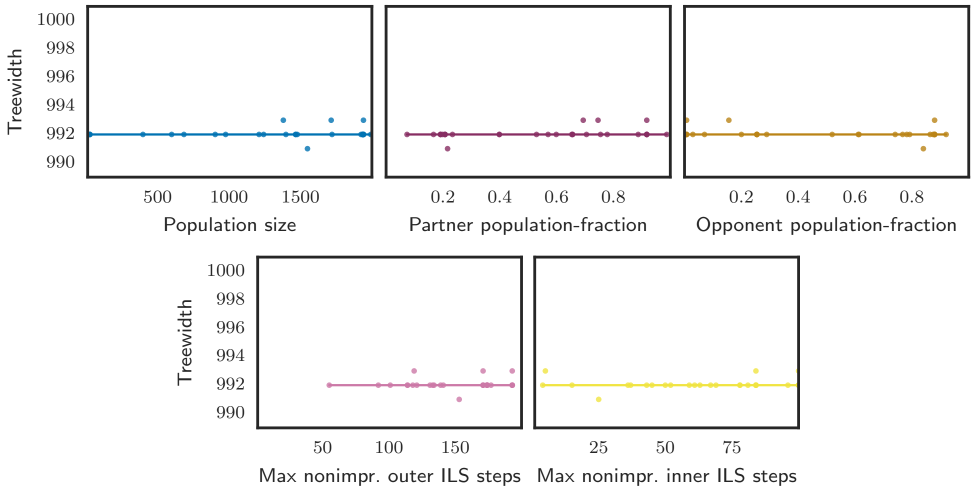


Figure 4.1: *MA1 applied to* `DSJC1000.9`, *31 samples.*

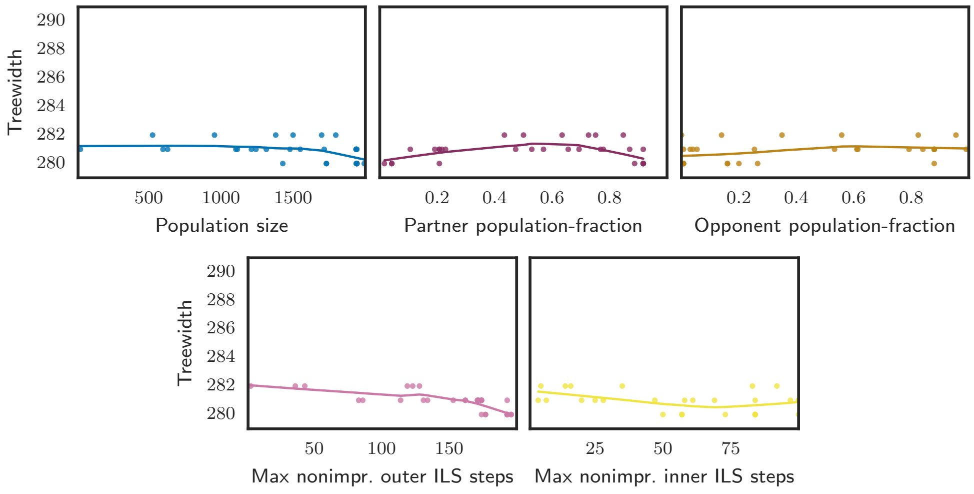


Figure 4.2: *MA1 applied to* `flat300_26_0`, *30 samples.*

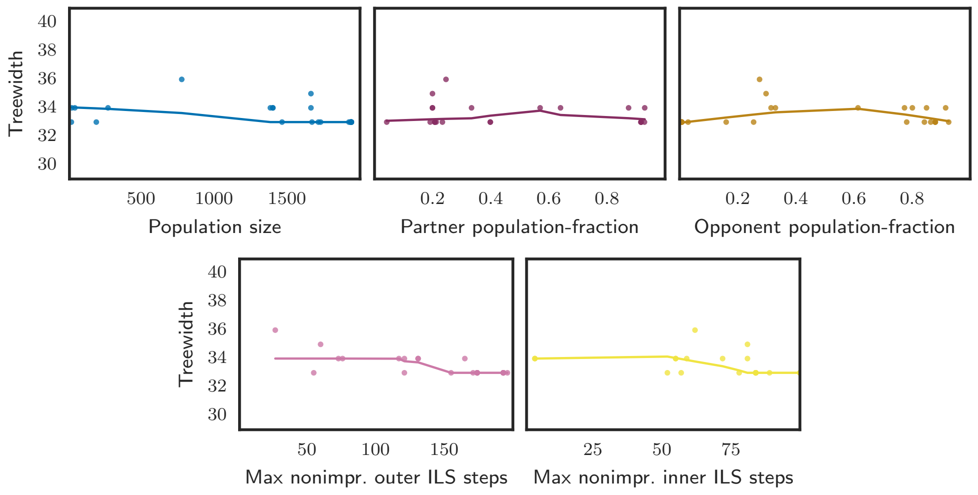


Figure 4.3: *MA1 applied to* `fpsol2.i.2`*, 24 samples.*

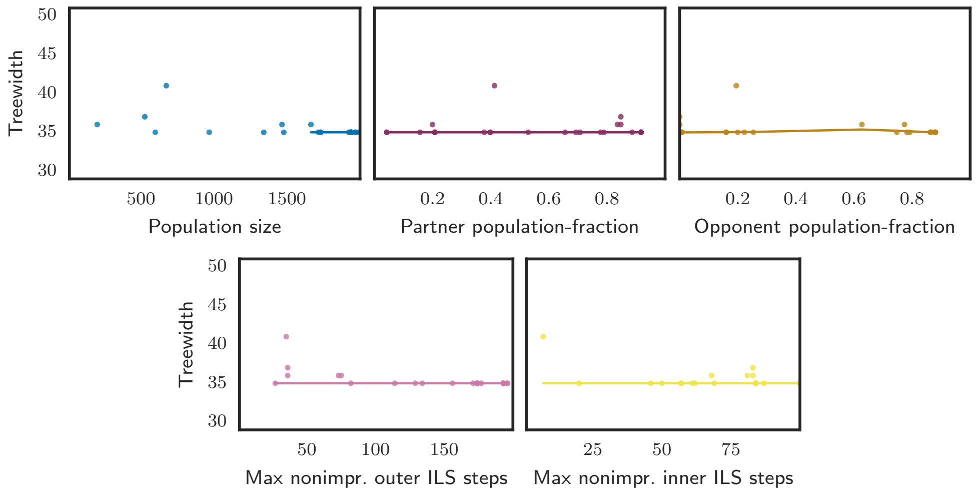


Figure 4.4: *MA1 applied to* `inithx.i.3`*, 25 samples.*

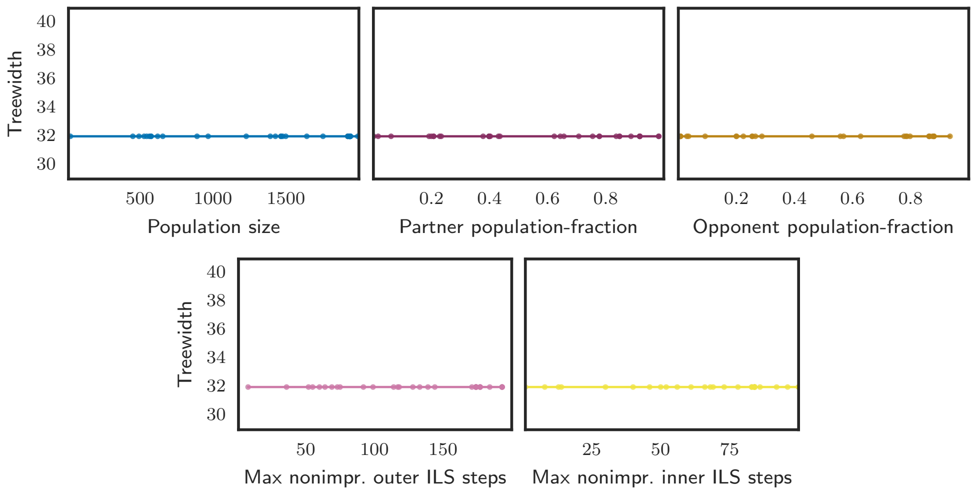


Figure 4.5: *MA1 applied to* `mulsol.i.2`*, 33 samples.*

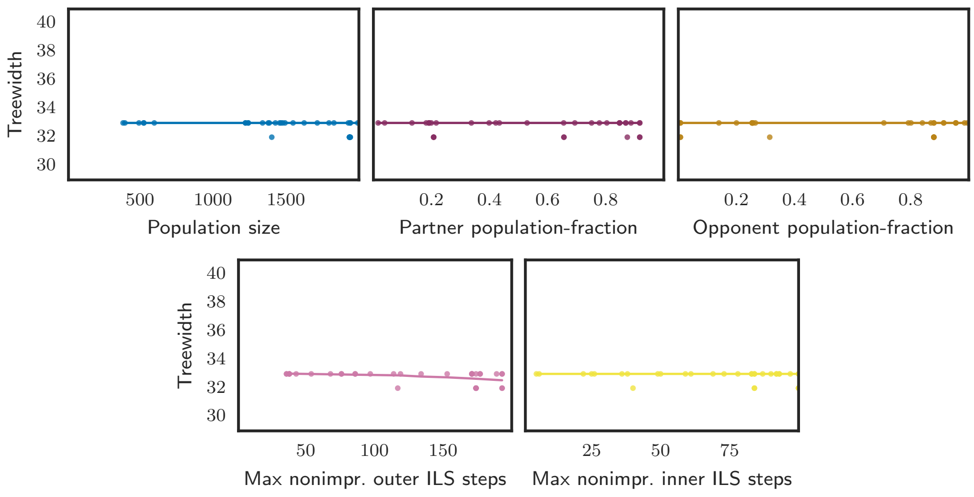


Figure 4.6: *MA1 applied to* `zeroin.i.3`*, 35 samples.*

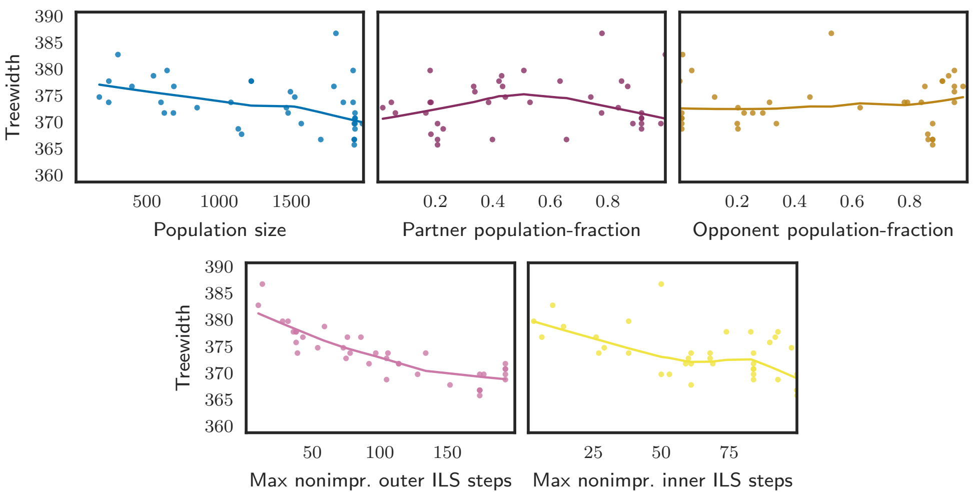


Figure 4.7: *MA1 applied to* `le450_15c`*, 38 samples.*

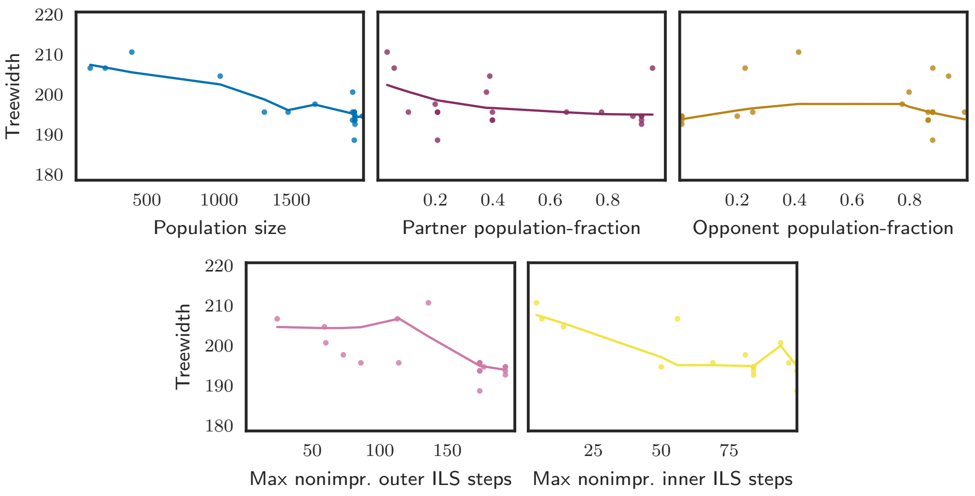


Figure 4.8: *MA1 applied to* `school1`*, 20 samples.*

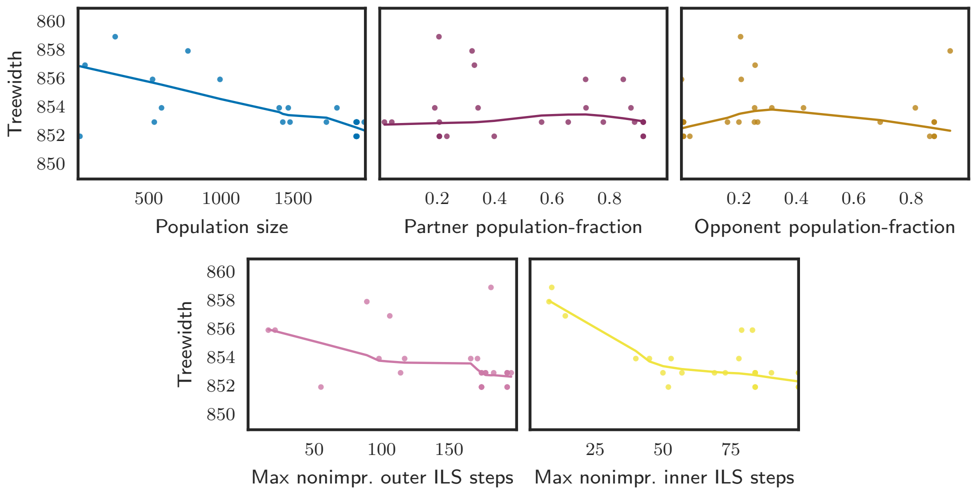


Figure 4.9: *MA1 applied to* `latin_square_10`*, 25 samples.*

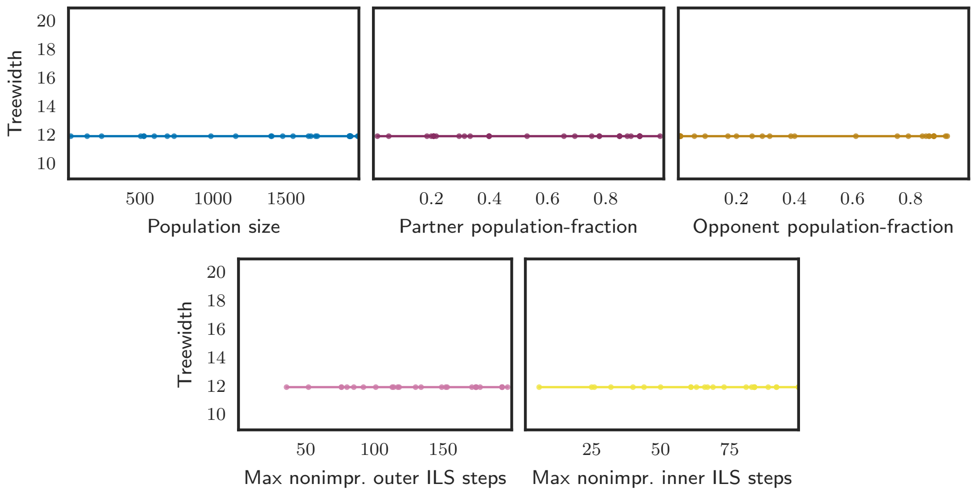


Figure 4.10: *MA1 applied to* `anna`*, 31 samples.*

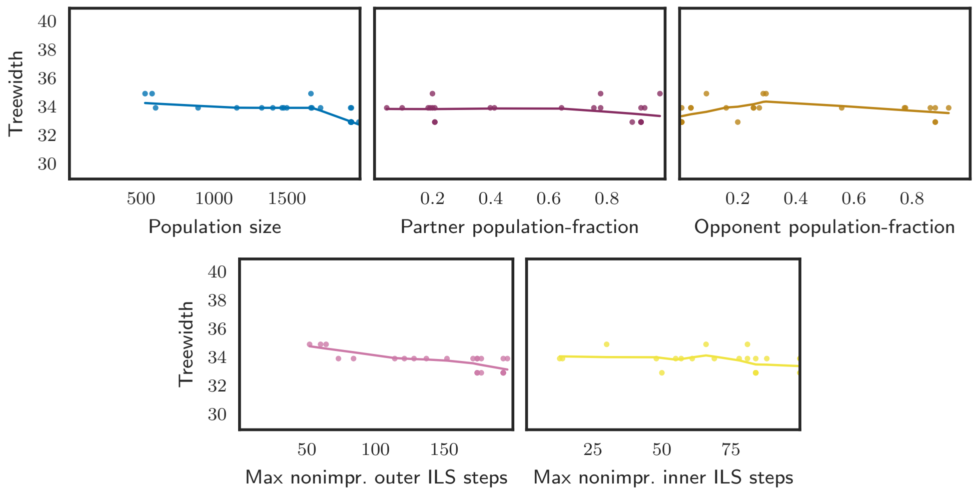


Figure 4.11: *MA1 applied to* `games120`*, 23 samples.*

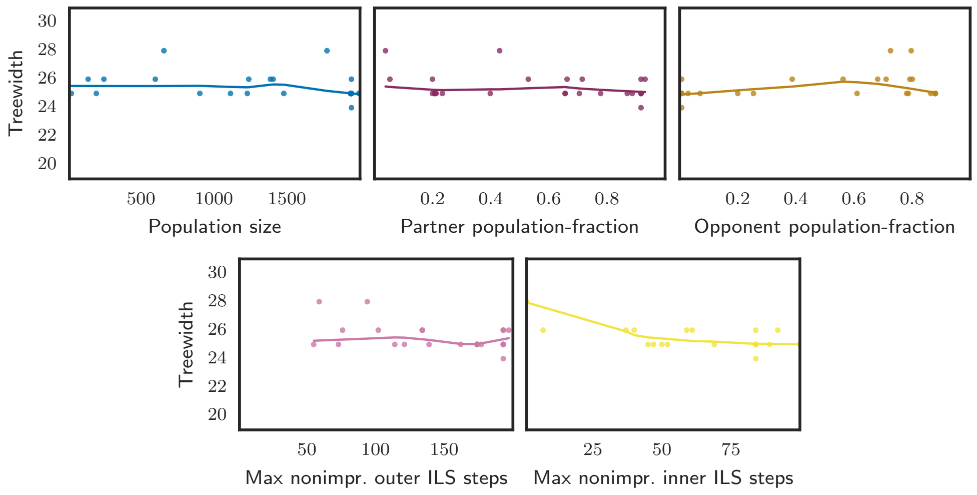


Figure 4.12: *MA1 applied to* `miles500`*, 26 samples.*

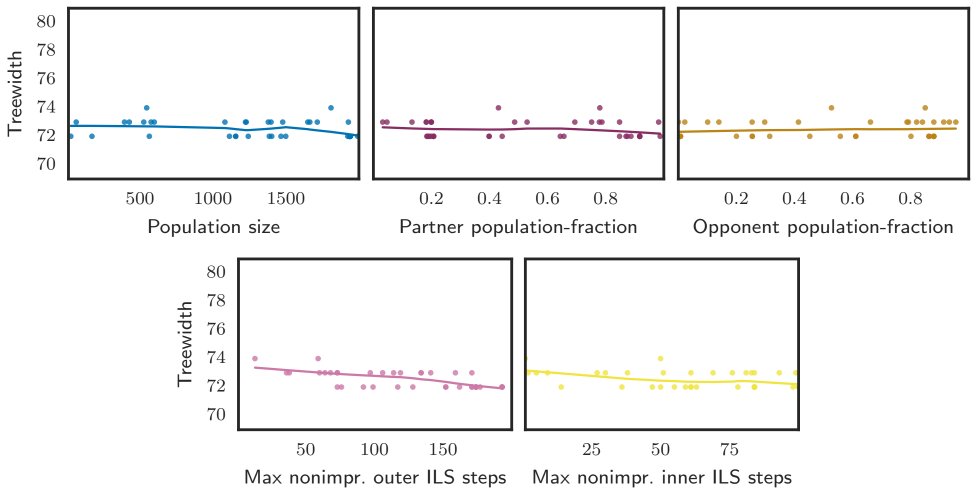


Figure 4.13: *MA1 applied to* `queen10_10`*, 41 samples.*

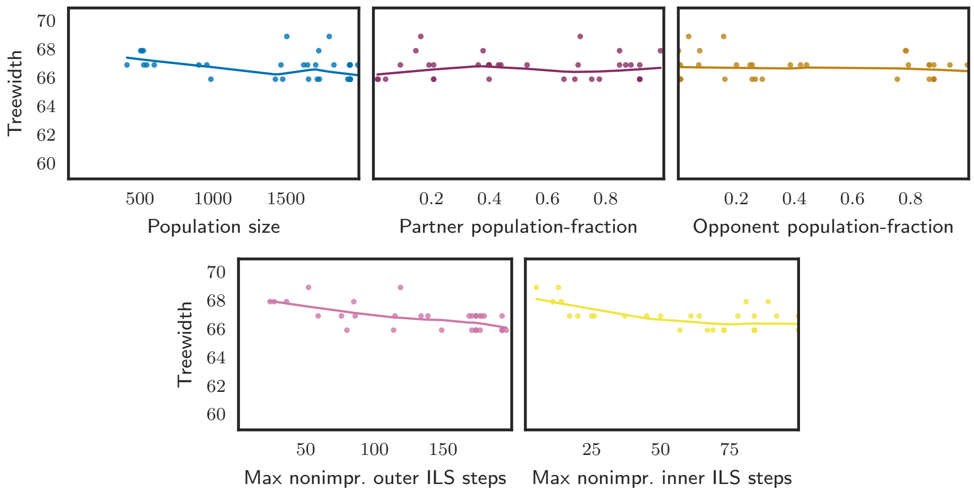


Figure 4.14: *MA1 applied to* `myciel7`*, 35 samples.*

4.2.2 MA2

The correlation plots, which are shown in Figures 4.15 to 4.28 on pages 43–49, do not, for the most part, show any big differences in the impact the individual parameters have on the outcome of the algorithm. Notable exceptions are the instances `le450_5c` (Figure 4.21), `school1_nsh` (Figure 4.22), `latin_square_10` (Figure 4.23), and `games120` (Figure 4.25), where the plots clearly affirm SMAC's results. Interestingly, the crossover parameter, for most instances, does not seem to have as much effect as initially expected.

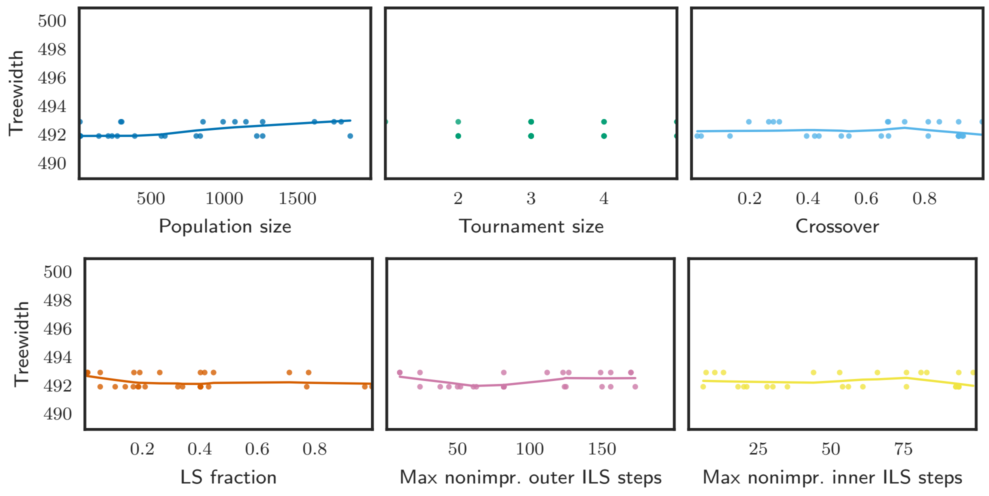


Figure 4.15: *MA2 applied to* `DSJC500.9`*, 31 samples.*

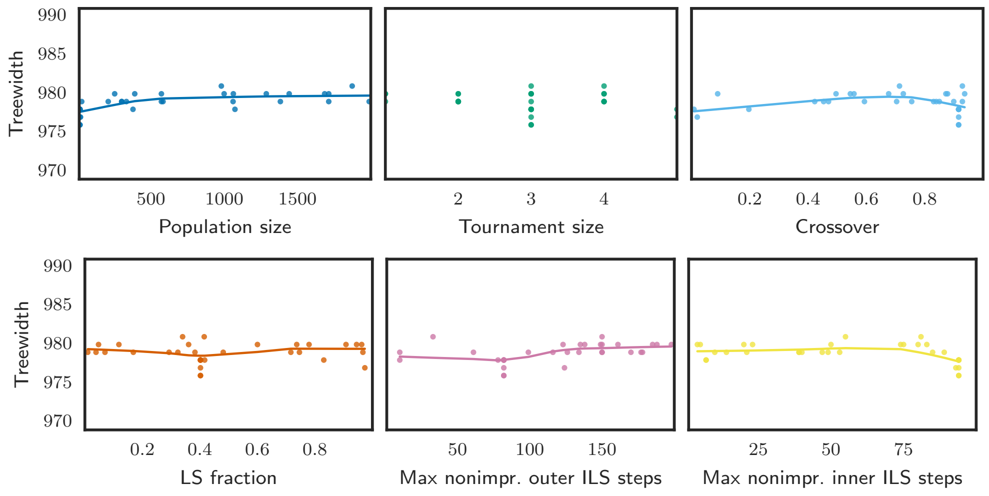


Figure 4.16: *MA2 applied to* `flat1000_76_0`*, 32 samples.*

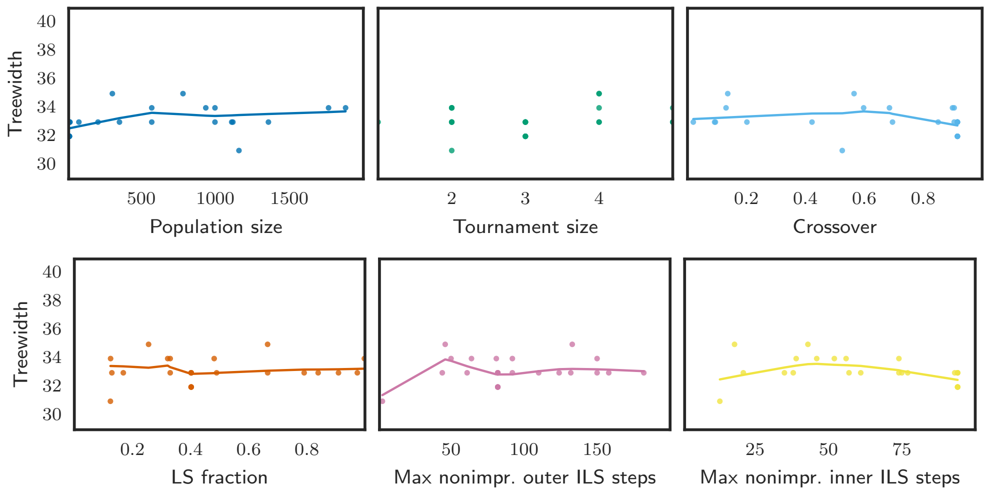


Figure 4.17: *MA2 applied to* `fpsol2.i.3`*, 24 samples.*

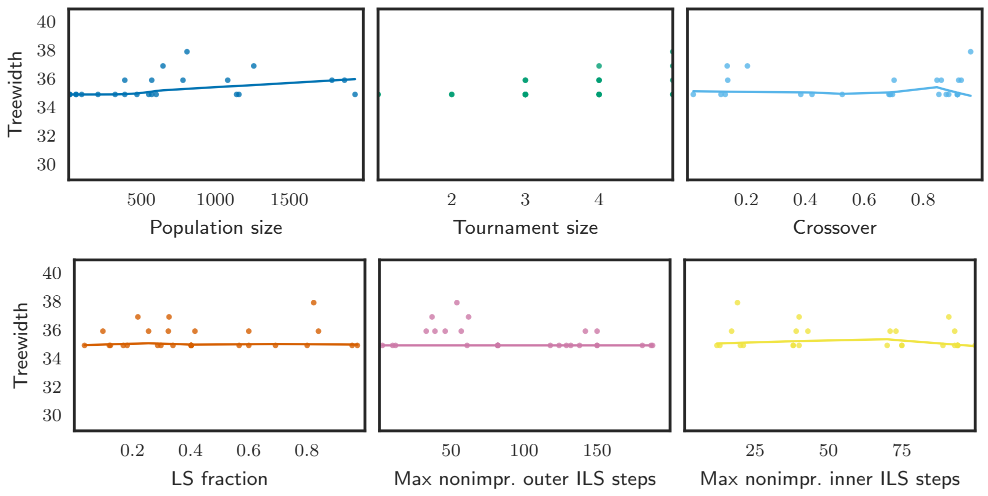


Figure 4.18: *MA2 applied to* `inithx.i.3`*, 30 samples.*

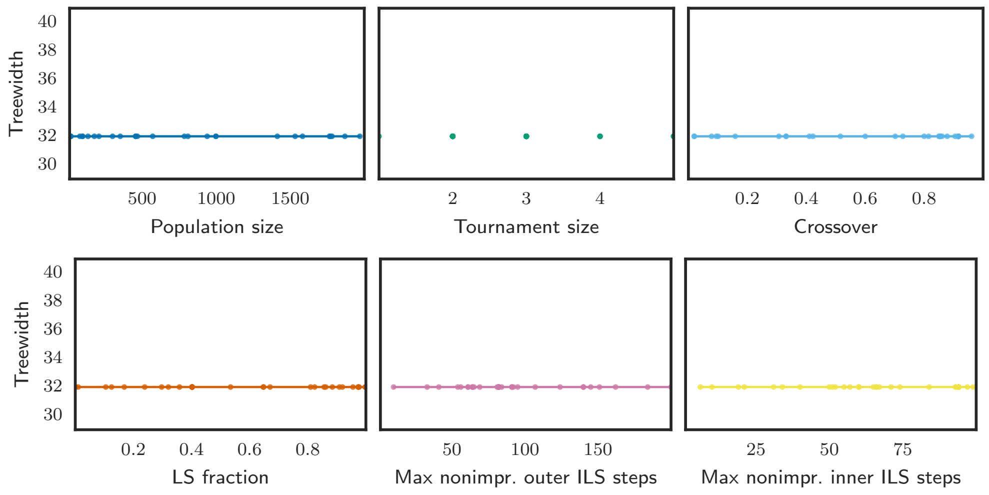


Figure 4.19: *MA2 applied to* `mulsol.i.2`*, 33 samples.*

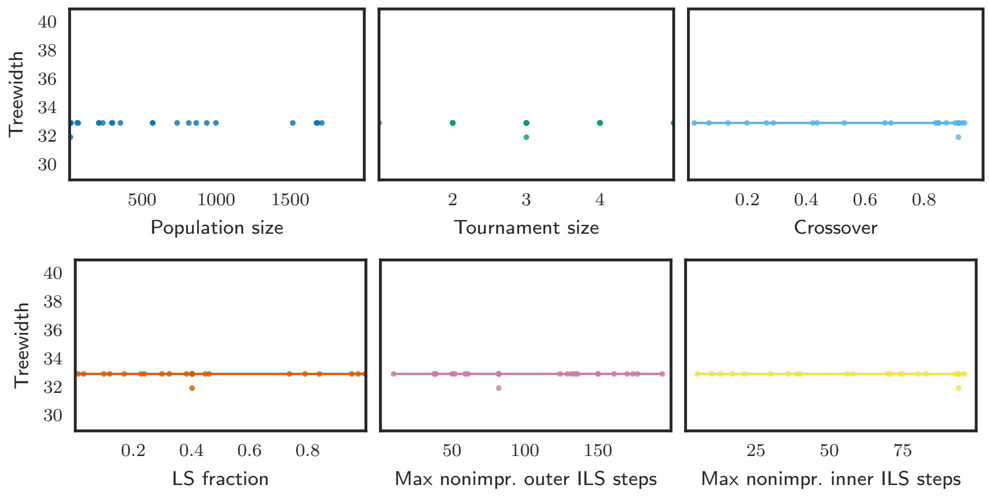


Figure 4.20: *MA2 applied to* `zeroin.i.3`*, 27 samples.*

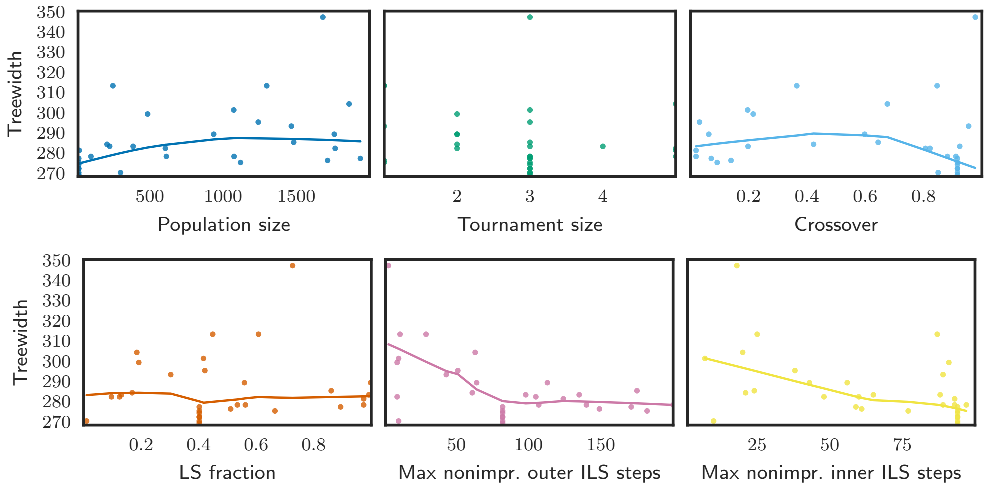


Figure 4.21: *MA2 applied to* `le450_5c`*, 31 samples.*

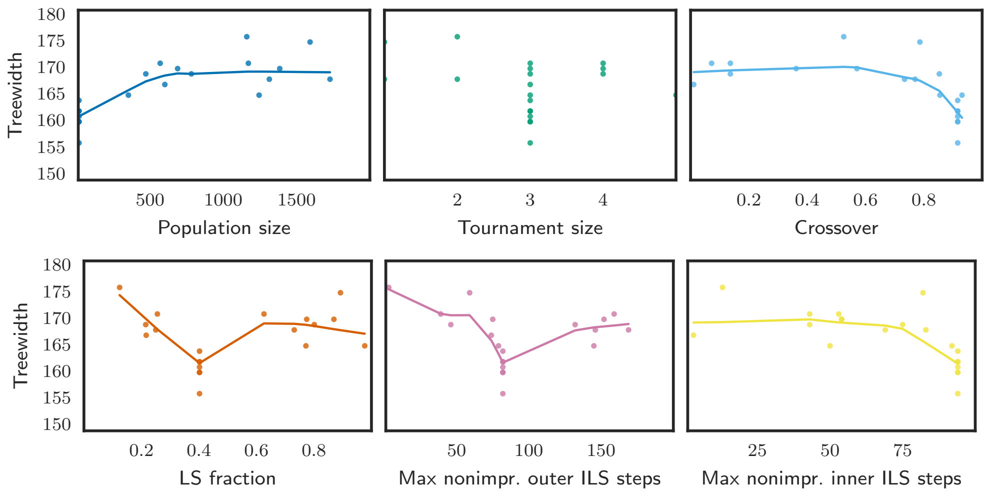


Figure 4.22: *MA2 applied to* `school1_nsh`*, 20 samples.*

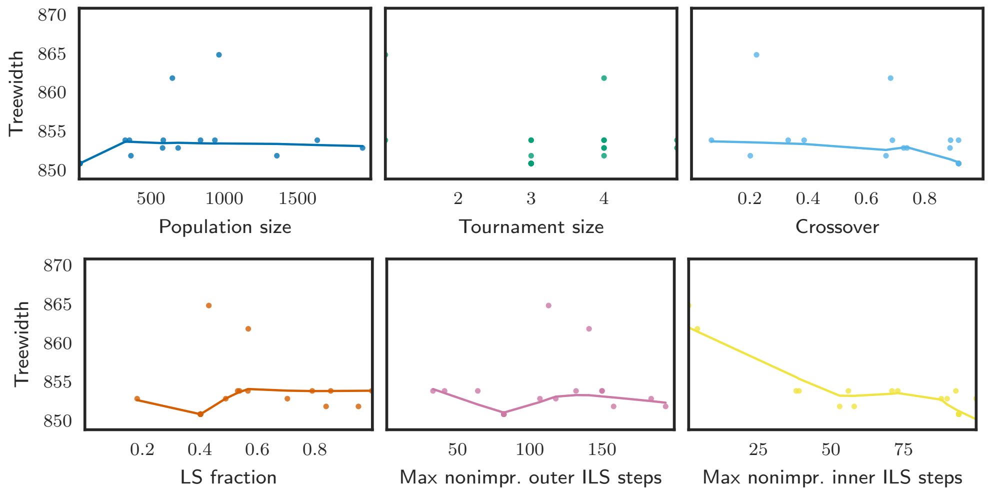


Figure 4.23: *MA2 applied to* `latin_square_10`*, 20 samples.*

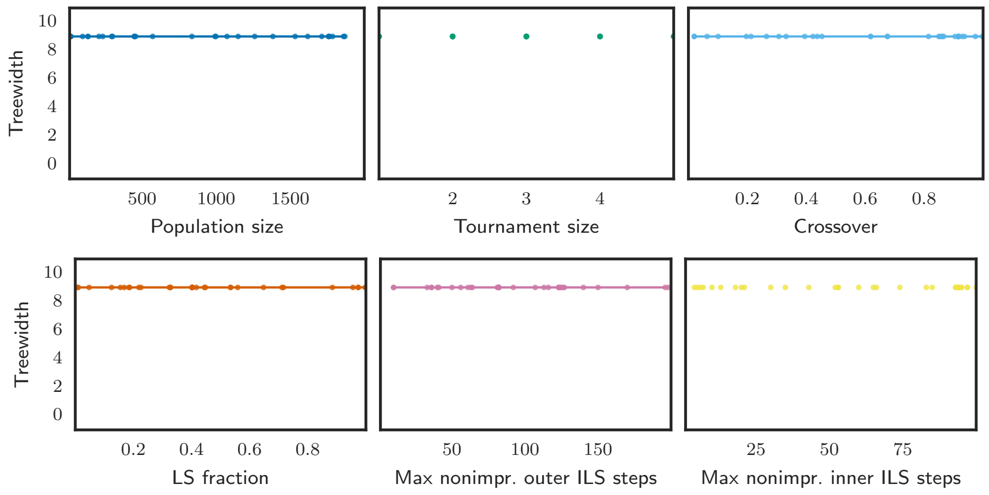


Figure 4.24: *MA2 applied to* `jean`*, 35 samples.*

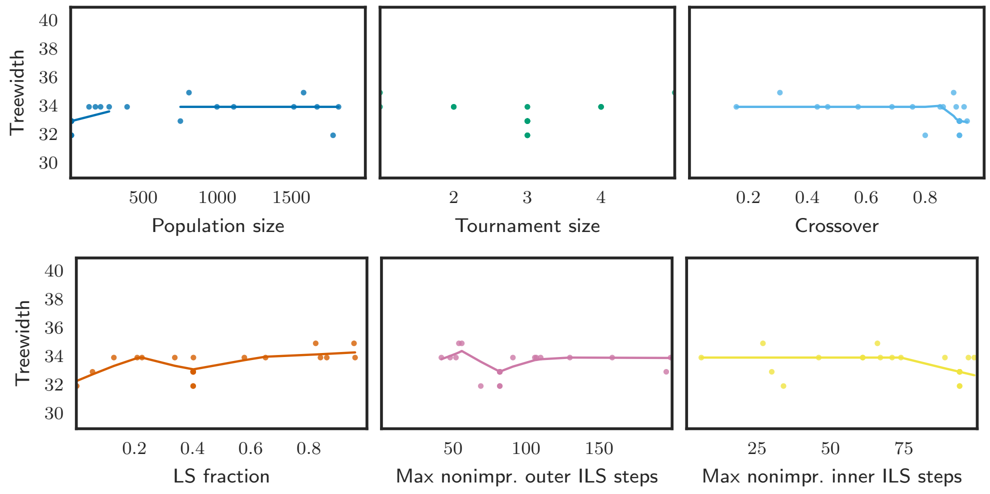


Figure 4.25: *MA2 applied to* `games120`*, 22 samples.*

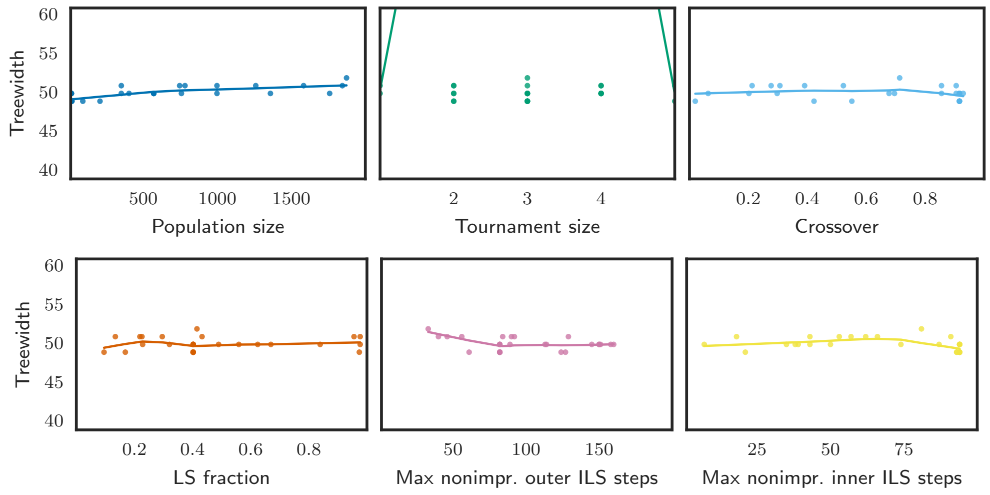


Figure 4.26: *MA2 applied to* `miles1000`*, 28 samples.*

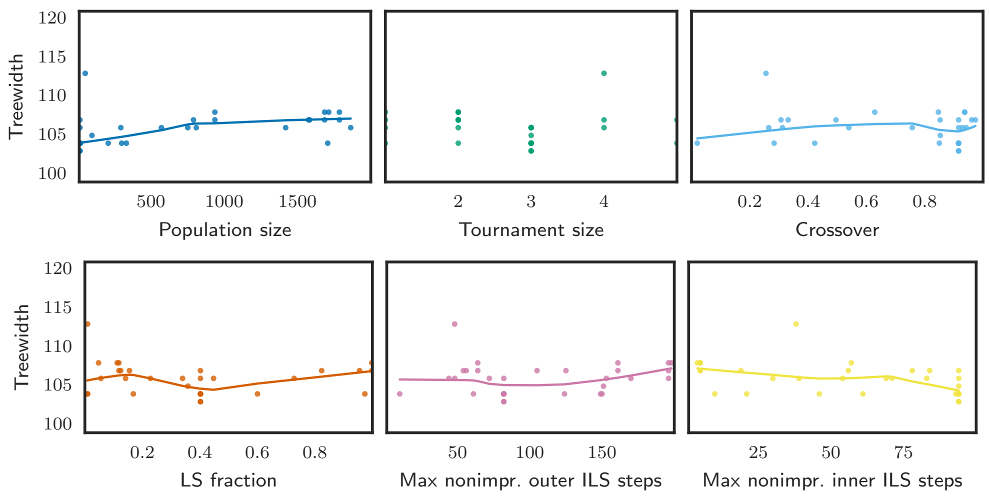


Figure 4.27: *MA2 applied to* `queen12_12`*, 31 samples.*

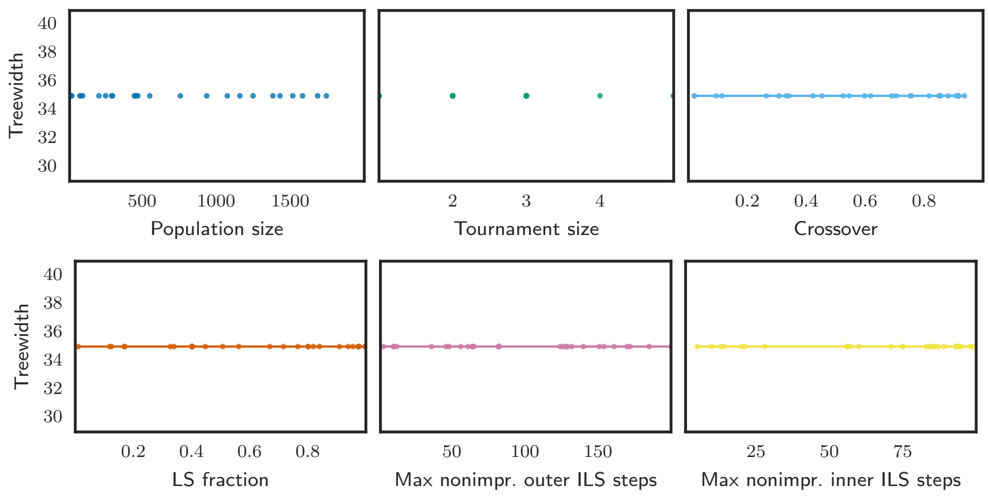


Figure 4.28: *MA2 applied to* `myciel6`*, 32 samples.*

4.2.3 MA3

Correlation plots for MA3 are shown in Figures 4.29 to 4.42 on pages 50–53.

For "simple" instances like `mulsol.i.1`, `zeroin.i.3`, `jean`, or `myciel6`, it seems that the parameter settings do not influence the result at all. The plots for the other instances show more variety. The population size parameter seems to induce good performance when chosen to be either low or high, but appears to have a negative impact over the remaining part of the spectrum. Lower is better seems to be true for the maximal number of nonimproving outer steps, albeit there is a trend for higher values to again improve results a bit.

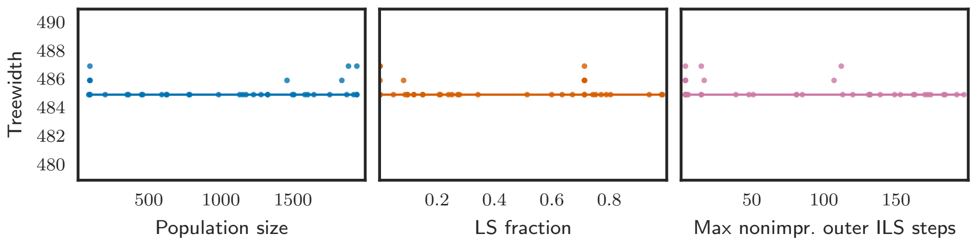


Figure 4.29: *MA3 applied to* `DSJR500.1c`*, 39 samples.*

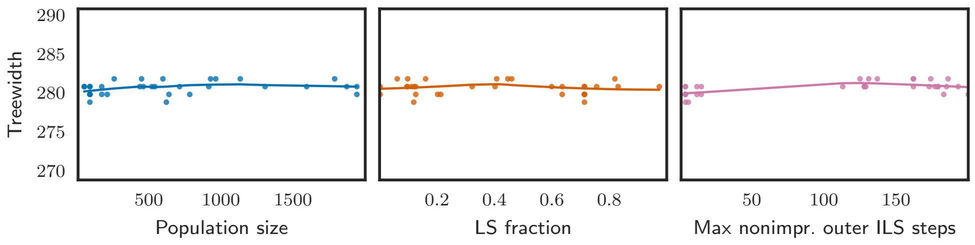


Figure 4.30: *MA3 applied to* `flat300_26_0`*, 32 samples.*

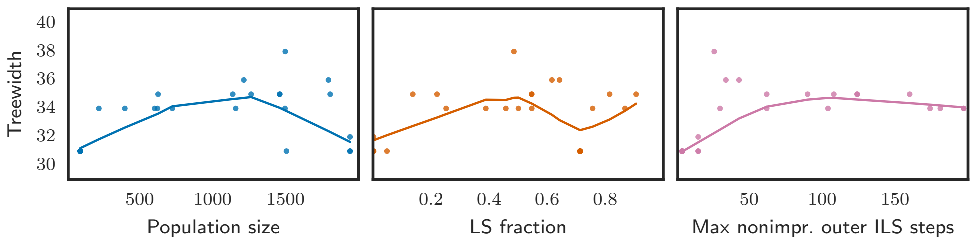


Figure 4.31: *MA3 applied to* `fpsol2.i.3`*, 27 samples.*

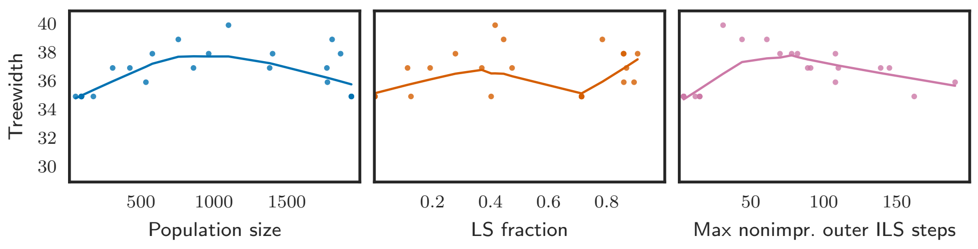


Figure 4.32: *MA3 applied to* `inithx.i.3`*, 25 samples.*

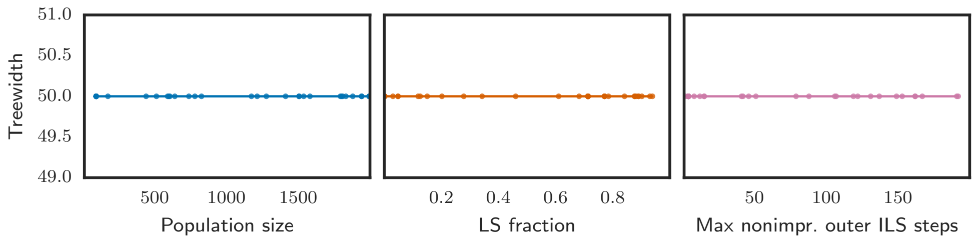


Figure 4.33: *MA3 applied to* `mulsol.i.1`*, 33 samples.*

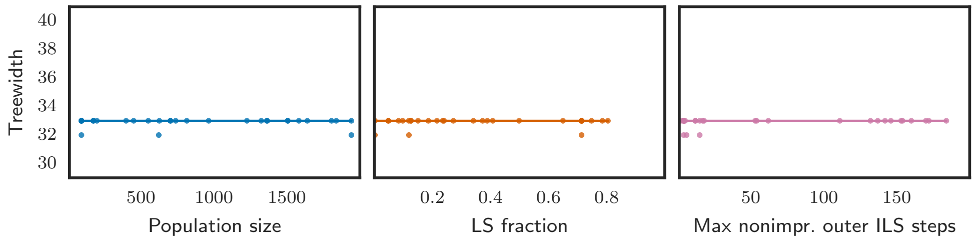


Figure 4.34: *MA3 applied to* `zeroin.i.3`*, 31 samples.*

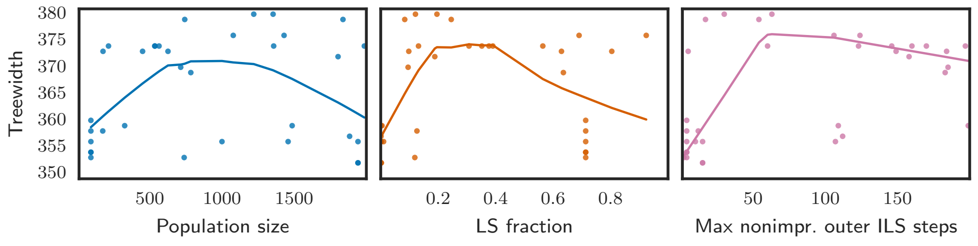


Figure 4.35: *MA3 applied to* `le450_15c`*, 34 samples.*

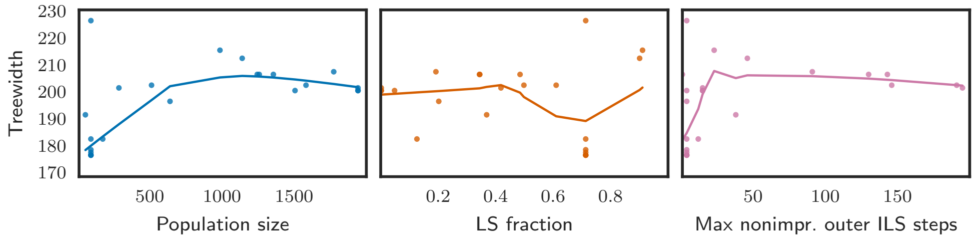


Figure 4.36: *MA3 applied to* `school1`*, 23 samples.*

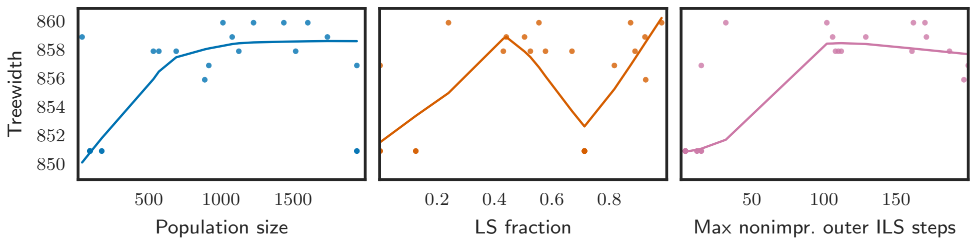


Figure 4.37: *MA3 applied to* `latin_square_10`*, 26 samples.*

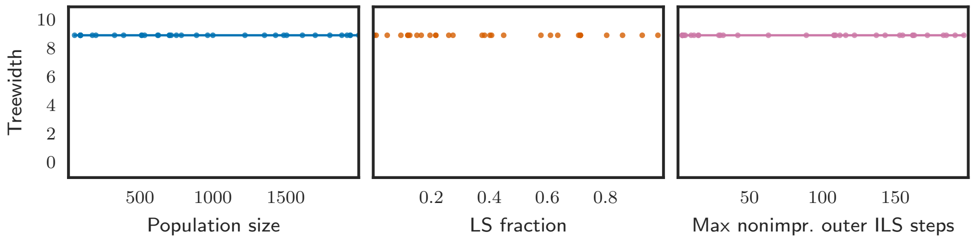


Figure 4.38: *MA3 applied to* `jean`*, 37 samples.*

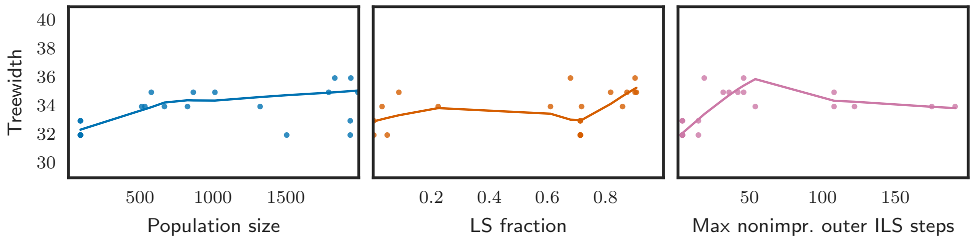


Figure 4.39: *MA3 applied to* `games120`*, 21 samples.*

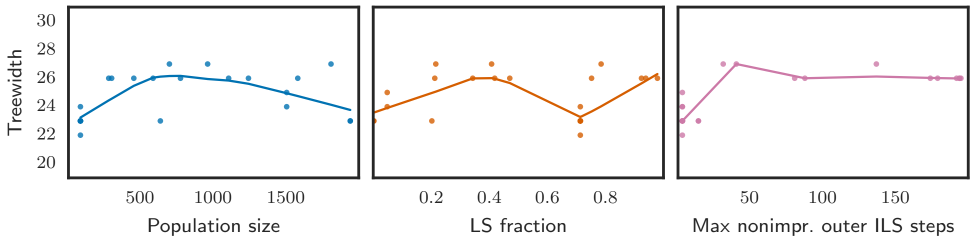


Figure 4.40: *MA3 applied to* `miles500`*, 23 samples.*

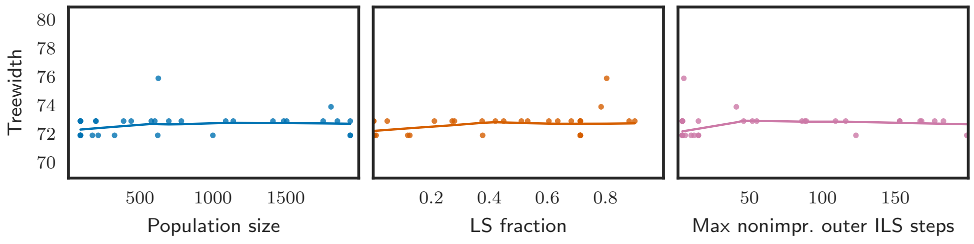


Figure 4.41: *MA3 applied to* `queen10_10`*, 32 samples.*

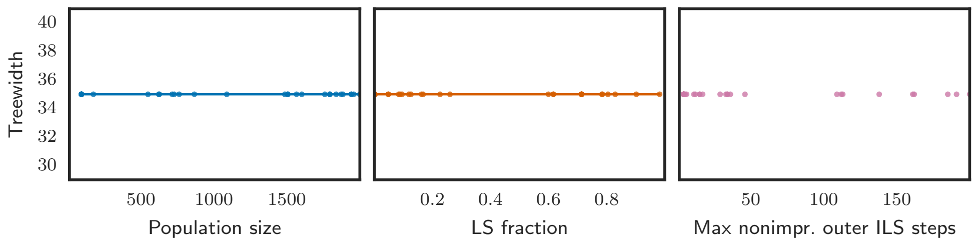


Figure 4.42: *MA3 applied to* `myciel6`*, 30 samples.*

Chapter 5

Comparative Experimental Evaluation

In this chapter we evaluate the performance of the new memetic algorithms by comparing them to state-of-the-art algorithms. In Section 5.1, the results on the benchmark instances from the Second DIMACS Implementation Challenge, which has been used as the training set for parameter tuning, are presented. The results on the validation instances are examined in Section 5.2.

5.1 Results on DIMACS Benchmark Instances

The benchmark instances of the Second DIMACS Implementation Challenge are often used to judge the performance of treewidth heuristics. Consequently, we utilize them as well, in order to show how our memetic algorithms compare to state-of-the-art algorithms in the field:

- Branch-and-bound algorithm for treewidth by Bachoore and Bodlaender (BB-tw) [6]
- Branch-and-bound algorithm for treewidth by Gogate and Dechter (BB-tw) [21]
- Tabu search for treewidth (TabuTW) [16]
- Ant colony system in combination with iterated local search for treewidth (ACS+ILS) [23, 24]
- Genetic algorithm for treewidth (GA-tw) [53, 46]
- Iterative heuristic algorithm for treewidth (IHA) [45]

Please note that all involved algorithms have been tuned specifically for this instance set. Therefore, the results do not necessarily apply to other instances (see Section 5.2).

Using their original source code, the results for IHA and GA-tw have been gathered on the same platform and with the same time limit as for the memetic algorithms. However, the results for the other algorithms are presented using the measurements from their respective papers; they have been executed on different machines, using different running times. Consequently, the comparison has to be taken with a grain of salt in this regard. Note that also IHA and GA-tw improve some upper bounds with respect to what has been reported for them originally. This is due to a faster computing platform, compared to what was used for the original experiments.

The algorithm runtime has been limited to one hour. For MA1, MA2, MA3, IHA, and GA-tw, the given numbers represent the best upper bounds from 20 runs per algorithm and instance.

Table 5.1 displays the result. Upper bounds that are equal to the best result are typed in boldface. Improvements over the previously best known upper bounds are highlighted. It is shown that the memetic algorithms perform well, and are able to improve upper bounds for 8 instances.

The Table shows that MA3 achieves better results than TabuTW and ACS+ILS on many instances, although, as mentioned above, the results for the latter have been obtained using different hardware.

IHA and GA-tw both find good results for most instances. Additionally, we had access to their source code. Because of that, the experiments on the validation set were conducted using only these two heuristics as competitors to our approaches.

Table 5.1: *Results on the benchmark instances of the Second DIMACS Implementation Challenge.*

	$\lvert V\rvert$	$\lvert E\rvert$	**MA1**	**MA2**	**MA3**	**IHA**	**GA-tw**	**TabuTW**	**ACS+ILS**	**BB-tw**	**QuickBB**
myciel3	11	20	**5**	**5**	**5**	**5**	**5**	**5**	**5**	-	**5**
myciel4	23	71	**10**	**10**	**10**	**10**	**10**	**10**	**10**	-	**10**
queen5_5	25	320	**18**	**18**	**18**	**18**	**18**	**18**	**18**	-	**18**
queen6_6	36	580	**25**	**25**	**25**	**25**	**25**	**25**	**25**	-	**25**
myciel5	47	236	**19**	**19**	**19**	**19**	**19**	**19**	**19**	-	**19**
queen7_7	49	952	**35**	**35**	**35**	**35**	**35**	**35**	**35**	-	**35**
queen8_8	64	1456	**45**	**45**	**45**	**45**	**45**	46	46	-	46
huck	74	602	**10**	**10**	**10**	**10**	**10**	**10**	**10**	-	**10**
jean	80	508	**9**	**9**	**9**	**9**	**9**	**9**	**9**	-	**9**
queen9_9	81	2112	**58**	**58**	**58**	**58**	**58**	**58**	59	-	59
david	87	812	**13**	**13**	**13**	**13**	**13**	**13**	**13**	**13**	**13**
myciel6	95	755	**35**	**35**	**35**	**35**	**35**	**35**	**35**	**35**	**35**
queen8_12	96	2736	**65**	**65**	**65**	**65**	**65**	-	-	-	-
queen10_10	100	2940	**72**	**72**	**72**	**72**	**72**	**72**	73	-	**72**
games120	120	1276	33	**32**	**32**	**32**	**32**	33	37	38	-
queen11_11	121	3960	**87**	**87**	**87**	**87**	**87**	88	89	-	89
DSJC125.1	125	1472	61	**60**	**60**	**60**	**60**	65	63	64	64

Table 5.1: *(continued) Results on the benchmark instances of the Second DIMACS Implementation Challenge.*

	$\lvert V\rvert$	$\lvert E\rvert$	*MA1*	*MA2*	*MA3*	*IHA*	*GA-tw*	*TabuTW*	*ACS+ILS*	*BB-tw*	*QuickBB*
DSJC125.5	125	7782	**108**	**108**	**108**	**108**	**108**	109	**108**	109	109
DSJC125.9	125	13 922	**119**	**119**	**119**	**119**	**119**	**119**	**119**	-	**119**
miles1000	128	6432	**49**	**49**	**49**	**49**	**49**	**49**	50	-	**49**
miles1500	128	10 396	**77**	**77**	**77**	**77**	**77**	**77**	**77**	-	**77**
miles250	128	774	**9**	**9**	**9**	**9**	10	**9**	**9**	-	**9**
miles500	128	2340	24	23	**22**	**22**	24	**22**	25	-	**22**
miles750	128	4226	37	**36**	**36**	**36**	37	**36**	38	-	37
anna	138	986	**12**	**12**	**12**	**12**	**12**	**12**	**12**	**12**	**12**
queen12_12	144	5192	104	**103**	**103**	**103**	104	104	109	-	110
queen13_13	169	6656	122	**121**	122	**121**	**121**	122	128	-	125
mulsol.i.3	184	3916	**32**	**32**	**32**	**32**	**32**	**32**	**32**	-	**32**
mulsol.i.4	185	3946	**32**	**32**	**32**	**32**	**32**	**32**	**32**	-	**32**
mulsol.i.5	186	3973	**31**	**31**	**31**	**31**	**31**	**31**	**31**	-	**31**
mulsol.i.2	188	3885	**32**	**32**	**32**	**32**	**32**	**32**	**32**	-	**32**
myciel7	191	2360	66	66	66	66	66	66	66	66	**54**
queen14_14	196	8372	142	141	142	**140**	**140**	141	150	-	143
mulsol.i.1	197	3925	**50**	**50**	**50**	**50**	**50**	**50**	**50**	-	**50**
zeroin.i.3	206	3540	**32**	**32**	**32**	**32**	**32**	**32**	33	-	-
zeroin.i.1	211	4100	**50**	**50**	**50**	**50**	**50**	**50**	**50**	-	-
zeroin.i.2	211	3541	**32**	**32**	**32**	**32**	**32**	**32**	33	-	-
queen15_15	225	10 360	164	162	**161**	162	**161**	163	174	-	167
DSJC250.1	250	6436	173	170	168	**167**	**167**	173	174	177	176
DSJC250.5	250	31 336	230	230	230	**229**	230	232	231	-	231
DSJC250.9	250	55 794	**243**	**243**	**243**	**243**	**243**	**243**	**243**	-	**243**
queen16_16	256	12 640	187	184	**183**	184	185	186	201	-	205
flat300_20_0	300	21 375	278	278	279	**277**	278	-	-	-	-
flat300_26_0	300	21 633	279	278	278	**277**	279	-	-	-	-
flat300_28_0	300	21 695	279	279	279	**278**	279	-	-	-	-
school1_nsh	352	14 612	159	153	**151**	152	156	162	185	-	-
school1	385	19 095	190	179	**176**	178	182	188	228	178	-
fpsol2.i.3	425	8688	32	**31**	**31**	**31**	**31**	**31**	**31**	-	**31**
le450_15a	450	8168	278	263	259	259	**258**	272	288	-	-
le450_15b	450	8169	275	264	262	**258**	264	270	292	-	289
le450_15c	450	16 680	362	355	351	351	**349**	359	368	-	372
le450_15d	450	16 750	365	355	352	**351**	353	360	371	-	371
le450_25a	450	8260	233	221	220	**216**	225	234	249	-	255
le450_25b	450	8263	231	216	**211**	214	222	233	245	-	251
le450_25c	450	17 343	334	323	**320**	322	**320**	327	346	-	349
le450_25d	450	17 425	336	330	**325**	326	326	336	355	-	349
le450_5a	450	5714	263	247	244	245	**243**	256	304	304	307
le450_5b	450	5734	256	248	**246**	**246**	247	254	308	-	309
le450_5c	450	9803	272	265	**263**	265	264	272	309	-	315
le450_5d	450	9757	269	263	**262**	265	265	278	290	-	303
fpsol2.i.2	451	8691	32	32	**31**	**31**	**31**	**31**	**31**	-	**31**
fpsol2.i.1	496	11 654	**66**	**66**	**66**	**66**	**66**	**66**	**66**	-	**66**
DSJC500.1	500	24 916	410	401	396	396	**395**	-	-	-	409

Table 5.1: *(continued) Results on the benchmark instances of the Second DIMACS Implementation Challenge.*

	$\lvert V\rvert$	$\lvert E\rvert$	MA1	MA2	MA3	IHA	GA-tw	TabuTW	ACS+ILS	BB-tw	QuickBB
DSJC500.5	500	125 248	**477**	478	**477**	**477**	**477**	-	-	-	479
DSJC500.9	500	224 874	**492**	**492**	**492**	**492**	**492**	-	-	-	**492**
DSJR500.1	500	7110	43	35	**33**	35	34	-	-	-	-
DSJR500.1c	500	242 550	**485**	**485**	**485**	**485**	**485**	-	-	-	**485**
DSJR500.5	500	117 724	263	250	246	250	258	-	-	-	**175**
homer	561	3258	29	**27**	**27**	31	31	31	30	-	31
inithx.i.3	621	13 969	35	35	35	35	35	35	**31**	35	**31**
inithx.i.2	645	13 979	35	35	35	35	35	35	**31**	35	**31**
inithx.i.1	864	18 707	**56**	**56**	**56**	**56**	**56**	**56**	**56**	-	**56**
latin[1]	900	307 350	**851**	**851**	**851**	**851**	**851**	-	-	-	-
DSJC1000.1	1000	99 258	901	891	882	882	**874**	-	-	-	896
DSJC1000.5	1000	499 652	978	975	976	**974**	975	-	-	-	977
DSJC1000.9	1000	898 898	**991**	992	992	**991**	992	-	-	-	**991**
flat1000_50_0	1000	245 000	978	976	975	**974**	975	-	-	-	-
flat1000_60_0	1000	245 830	978	976	976	**975**	**975**	-	-	-	-
flat1000_76_0	1000	246 708	978	976	976	**974**	975	-	-	-	-

5.2 Algorithm Validation

The algorithms are validated on a set of instances that is deliberately distinct from the training set used in the previous chapter. This validation set, which is listed in Tables 5.2 on the next page and 5.3 on page 60, has been taken from TreewidthLib [13], a website devoted to treewidth benchmark instances (and algorithms), created by Jan-Willem van den Broek and Hans Bodlaender. Using two distinct sets of instances ensures that we are able to improve best known upper bounds on the training instances by specifically tuning for them, and, at the same time, reason about the universal performance of the algorithms.

The validation runs for GA-tw and IHA have been executed with their respective original implementation. Merely minor modification have been applied, which were required for compilation and solution extraction.

By using the exact same instances for all involved algorithms, instance variability is factored out of the comparison [30] (this method is also known as *blocking on instances* [50]).

Due to the architectural differences in the algorithms, there is no natural metric for measuring their progress. Instead, a limit on their runtime is imposed. We chose a limit of one hour, which seems reasonable, considering that the involved algorithms are usually meant for preprocessing a problem instance before handing over the result to another algorithm that actually solves the original problem. By forbearing from longer running times, we were

[1]latin_square_10

Table 5.2: *The validation set, instances 1–12. The instance files and the corresponding descriptions have been taken from [13].*

	|**V**|	|**E**|	
`barley`	48	126	The first version of the Barley network, a probabilistic network developed by Kristensen and Rasmussen in a project for growing malting barley without use of pesticides. This is the moralized graph of the network.
`rl5934`	2048	3087	TSP Graphs from the work of Cook and Seymour.
`pcb3038`	1985	3109	
`fnl4461`	3326	5147	
`fl3795`	2103	3973	
`pathfinder`	109	211	A probabilistic network for assisting surgical pathologists with the diagnosis of lymph-node diseases. The graph given here is the undirected graph, obtained from applying moralization to the directed probabilistic network.
`oesoca+`	67	208	Probabilistic network for staging of oesophageal carcinoma, developed at Utrecht University by L. van der Gaag et al. The graph given here is the version after the moralization step.
`link`	724	1738	Probabilistic network for linkage analysis.
`diabetes`	413	819	A preliminary model for insulin dose adjustment that consists of 24 structurally identical subnetworks interconnected via temporal links. [4]
`pigs`	441	806	Pedigree of breeding pigs. The pedigree is used for diagnosing the PSE disease. Created by Claus S. Jensen on the basis of a data base from Søren Andersen (Danske Slagterier, Axeltorv Copenhagen).
`water`	32	123	Preliminary model of the biological processes of a water purificationplant. The network consists of four structurally identical subnetworks, each representing a time slice of 15 minutes. The network was developedby Finn V. Jensen, Uffe Kjærulff, Kristian G. Olesen, and Jan Pedersen. This is a probabilistic network, given here after the moralization step.
`munin1`	189	366	A subset of the Munin network; a probablistic network. The version here is after the moralization step.

able to gather 20 samples for each of the 5 validated algorithms, for each of the 106 instances (that includes the runs for Section 5.1), in a time of 10600 hours, or approximately 442 days. In other words, the one hour runtime limit and the sample size of 20 represent a compromise between the time needed for the validation, and its informative value.

Table 5.3: *The validation set, instances 13–27. The instance files and the corresponding descriptions have been taken from [13].*

	|V|	|E|	
1ubq	74	211	A graph from the field of computational biology, where tree decomposition directs a dynamic programming algorithm for protein redesign. The graph comes from ubiquitin, a well studied protein (PDB ID 1ubq). Each vertex represents a single side chain. Each edge represents the existence of a pairwise interaction between the two side chains. Copyright Andrew Leaver-Fay.
knights8_8	64	168	The vertices of this graph correspond to the squares of an 8 by 8 chessboard. There is an edge between vertices if these are a knights-move away from each other. The origin of this graph can be found in the more than 1000 years old knights-tour puzzle: to find a Hamiltonian path or cycle in this graph.
1a62	122	1516	
1sem	57	570	A graphs from the field of computational biology, where tree decomposition directs a dynamic programming algorithm for protein redesign. Each vertex represents a single side chain. Each edge represents the existence of a pairwise interaction between the two side chains. Copyright Andrew Leaver-Fay.
1qtn	87	788	
1pwt	61	657	
1or7	180	1875	
1on2	135	1527	
1oai	58	524	
BN_0	100	300	Moralized version of a graph used in the UAI 2006 Inference Evaluation. This is a random modification of the Alarm network.
BN_16	2127	5581	Moralized version of a graph used in the UAI 2006 Inference Evaluation. This is a modified form of the Diagnosis graph.
BN_20	2843	8108	Moralized version of a graph used in the UAI 2006 Inference Evaluation. This is a DBN from speech recognition that is unrolled a fixed amount.
BN_30	1156	3333	Moralized version of a DAG used in the UAI 2006 Inference Evaluation. The original DAG was a grid.
BN_42	880	1577	Moralized version of a DAG used in the UAI 2006 Inference Evaluation. The original DAG was from iscas85.
BN_47	661	2131	Moralized version of a DAG used in the UAI 2006 Inference Evaluation. The original DAG was from iscas89.

The experiments have been done on an AMD Opteron 6272 CPU, running at 2.1 GHz, with access to roughly 220 GB of memory. Since the CPU speed is the most important performance factor for this work's heuristic algorithms, the following result of the (randomly selected) *pystone* benchmark may be taken into consideration when reproducing the presented results:

```
$ python2 --version
Python 2.7.3
$ python2 -c "from test import pystone; pystone.main()"
Pystone(1.1) time for 50000 passes = 0.71
This machine benchmarks at 70422.5 pystones/second
```

5.2.1 Results

In the following the results for the validation set are presented. Those instances on which all tested algorithms perform very similarly are listed in Chapter A on page 85.

rl5934

Table 5.4: *Results for instance* rl5934.

	Best		Average	
	treewidth	**seconds**	**treewidth**	**samples**
MA1	41	3608.1	45.1 ± 2.3	20
MA2	36	3600.1	40.6 ± 2.6	20
MA3	24	3316.0	33.6 ± 6.9	20
IHA	27	1032.3	33.0 ± 3.6	20
GAtw	21	2154.5	27.0 ± 4.4	20

Table 5.4 shows the algorithm results, including the sample means and standard deviations. Noticeably, with a value of 6.9, MA3's standard deviation is almost twice as high as IHA's.

Figure 5.1 on the next page shows a violin plot of the runs against this instance. A violin plot combines a boxplot with a kernel density estimate, which helps us to visually compare the results. In addition to the sample mean, the 25th and 75th percentiles are also shown in the plots.

The data suggests that even though MA3 might be able to produce good results, the quality of its results varies heavily. In order to tell whether the apparent differences in mean values across the different algorithms is statistically significant, we apply statistical tests.

The first candidate is the one-way analysis of variance (ANOVA), which requires the samples to be independent, normally distributed, and of equal variance. The independence is assured as the algorithms do not influence each

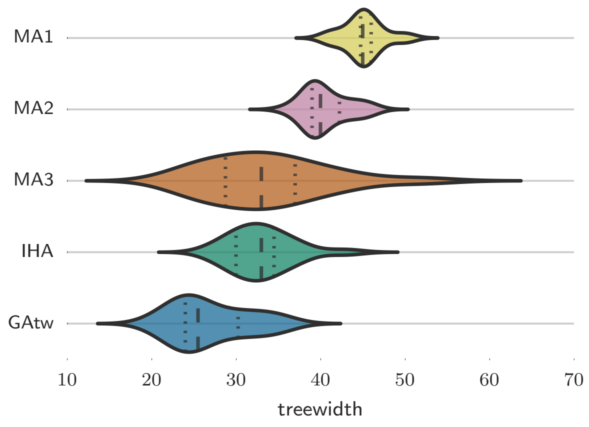


Figure 5.1: *Violin plot for instance* `rl5934`.

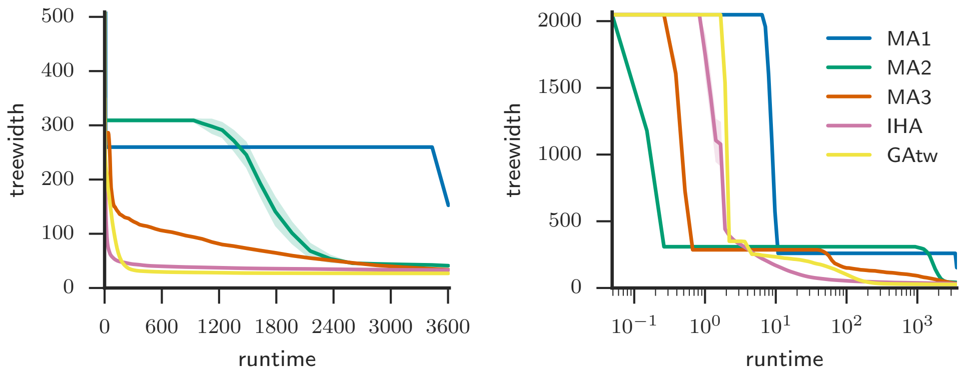


Figure 5.2: *Solution quality over time for instance* `rl5934` *(linear time scale left, logarithmic time scale right).*

other in any way. All five samples are distributed normally according to both the simple skewness test and the Shapiro-Wilk test, at significance levels of 5 % and 2 %, respectively. Their variance differs, however, as both the Bartlett test and the Levene test assure, at significance levels of 5 % each. Therefore, instead of the one-way ANOVA, the Kruskal-Wallis test is chosen, as it does not require homogeneity of variance.

The Kruskal-Wallis test rejects its null hypothesis, which means that at least two means are significantly different from each other. In order to find out which, we deploy two tests: Tukey's test and a pairwise Welch's t-test with the more conservative Bonferroni correction. Tukey's test assumes homogeneity of variance, which we do not have, but with equal sample sizes the test should be robust enough to compensate the variance differences (all other requirements are fulfilled). For the other test, the Bonferroni correction, which accounts for the multiple comparison, is calculated using $\alpha_{\text{local}} = 1 - (1 - \alpha_{\text{global}})^{1/n}$, where n is the number of tests we execute. With $m = 5$ variants there are $\frac{m(m-1)}{2} = 10$ possible pairwise comparisons; we omit IHA vs. GA-tw, so we have $n = 9$. Both tests have been done twice, for the global α values of 10 and 5. The results are shown in Table 5.5.

Table 5.5: *Pairwise comparison of means for instance* `rl5934`.

		Tukey's test		**Welch and Bonfer.**	
	mean diff.	$\alpha = \mathbf{10\,\%}$	$\alpha = \mathbf{5\,\%}$	$\alpha = \mathbf{10\,\%}$	$\alpha = \mathbf{5\,\%}$
IHA vs MA1	12.05	distinct	distinct	distinct	distinct
IHA vs MA2	7.5	distinct	distinct	distinct	distinct
IHA vs MA3	0.6	equal	equal	equal	equal
GA-tw vs MA1	18.15	distinct	distinct	distinct	distinct
GA-tw vs MA2	13.6	distinct	distinct	distinct	distinct
GA-tw vs MA3	6.7	distinct	distinct	distinct	distinct
MA1 vs MA2	4.55	distinct	distinct	distinct	distinct
MA1 vs MA3	11.45	distinct	distinct	distinct	distinct
MA2 vs MA3	6.9	distinct	distinct	distinct	distinct

So what we see in Figure 5.1 on the preceding page is evidently statistically significant. Consequently, on this instance, GA-tw dominates its competitors in terms of solution quality. Regarding the runtime behavior, which is displayed in Figure 5.2 on the facing page, GA-tw is still doing well, while MA1 performes badly, and also MA2 is quite slow on this instance.

`pcb3038`

Table 5.6: *Results for instance* `pcb3038`.

	Best		**Average**	
	treewidth	**seconds**	**treewidth**	**samples**
MA1	39	3607.6	42.2 ± 1.7	20
MA2	36	3498.2	38.9 ± 2.2	20
MA3	24	3456.9	28.9 ± 3.2	20
IHA	25	1859.1	29.6 ± 3.5	20
GAtw	19	2159.3	20.4 ± 0.7	20

The results for this instance look very similar to those for the previous one (see Table 5.6 and Figure 5.3 on the following page).

Since, according to the Shapiro-Wilk test, GA-tw cannot be assumed to be distributed normally, we use the Mann-Whitney U test, again with Bonferroni correction. Its results are shown in Table 5.7 on the next page.

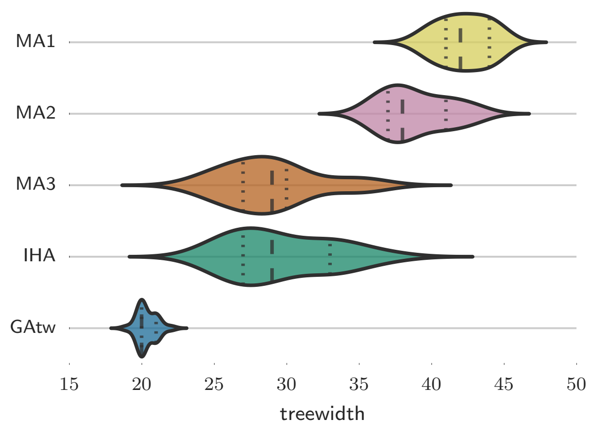


Figure 5.3: *Violin plot for instance* `pcb3038`*.*

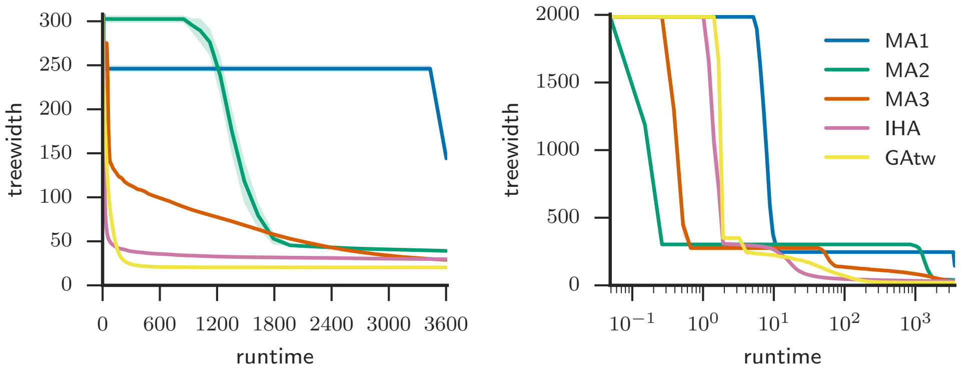


Figure 5.4: *Solution quality over time for instance* `pcb3038` *(linear time scale left, logarithmic time scale right).*

Table 5.7: *Pairwise comparison of means for instance* `pcb3038`*.*

		Mann-Whitney U test	
	mean diff.	$\alpha = \mathbf{10\,\%}$	$\alpha = \mathbf{5\,\%}$
IHA vs MA1	12.6	distinct	distinct
IHA vs MA2	9.25	distinct	distinct
IHA vs MA3	0.75	equal	equal
GA-tw vs MA1	21.9	distinct	distinct
GA-tw vs MA2	18.55	distinct	distinct
GA-tw vs MA3	8.55	distinct	distinct
MA1 vs MA2	3.35	distinct	distinct
MA1 vs MA3	13.35	distinct	distinct
MA2 vs MA3	10.0	distinct	distinct

fnl4461

Table 5.8: *Results for instance* `fnl4461`.

	Best		**Average**	
	treewidth	**seconds**	**treewidth**	**samples**
MA1	59	3619.9	67.6 ± 3.9	20
MA2	61	3600.2	71.0 ± 5.2	20
MA3	122	3505.5	146.8 ± 13.1	20
IHA	39	2980.7	48.6 ± 5.9	20
GAtw	31	2606.6	33.6 ± 2.3	20

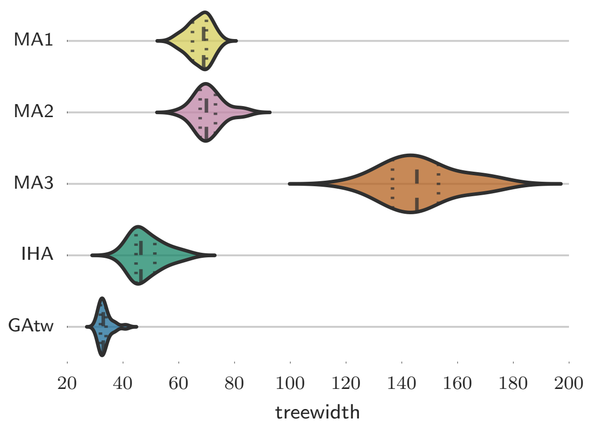


Figure 5.5: *Violin plot for instance* `fnl4461`.

The results for this instance show a great variety—the worst average result is more than four times as large as the best average result. Again, normal distribution cannot be assumed. The results of the pairwise Mann-Whitney U tests with Bonferroni correction are given in Table 5.9 on the following page.

Figure 5.6 on the next page, which shows the time response, shows that MA1 and MA2 are by far the slowest algorithms for this instance.

fl3795

The results given in Table 5.10 on the following page and Figure 5.7 on page 67 are obviously quite clearly separated. The results of the pairwise, Bonferroni-corrected Mann-Whitney U tests (Table 5.11 on page 67), which are done on the not normally distributed samples, confirm this observation.

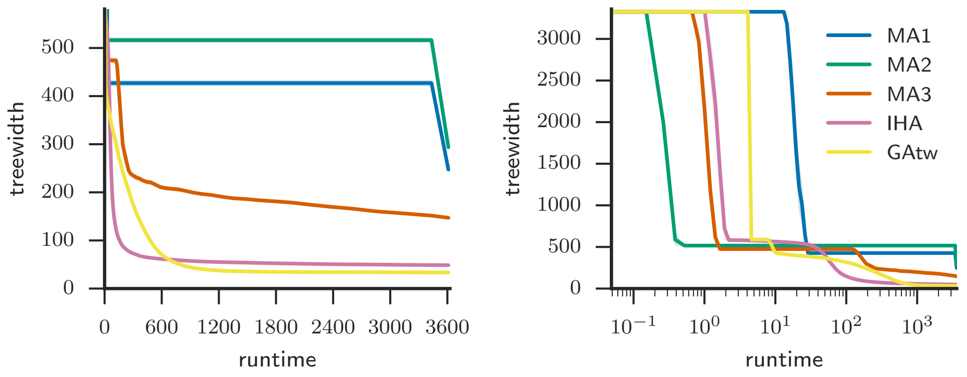


Figure 5.6: *Solution quality over time for instance* `fnl4461` *(linear time scale left, logarithmic time scale right).*

Table 5.9: *Pairwise comparison of means for instance* `fnl4461`.

		Mann-Whitney U test	
	mean diff.	$\alpha = \mathbf{10\,\%}$	$\alpha = \mathbf{5\,\%}$
IHA vs MA1	18.95	distinct	distinct
IHA vs MA2	22.3	distinct	distinct
IHA vs MA3	98.1	distinct	distinct
GA-tw vs MA1	34.05	distinct	distinct
GA-tw vs MA2	37.4	distinct	distinct
GA-tw vs MA3	113.2	distinct	distinct
MA1 vs MA2	3.35	equal	equal
MA1 vs MA3	79.15	distinct	distinct
MA2 vs MA3	75.8	distinct	distinct

Table 5.10: *Results for instance* `fl3795`.

	Best		**Average**	
	treewidth	**seconds**	**treewidth**	**samples**
MA1	30	3606.7	31.2 ± 0.8	20
MA2	27	2836.2	29.2 ± 1.3	20
MA3	21	2932.5	23.6 ± 1.7	20
IHA	18	1052.0	18.7 ± 0.6	20
GAtw	13	1910.8	13.7 ± 0.6	20

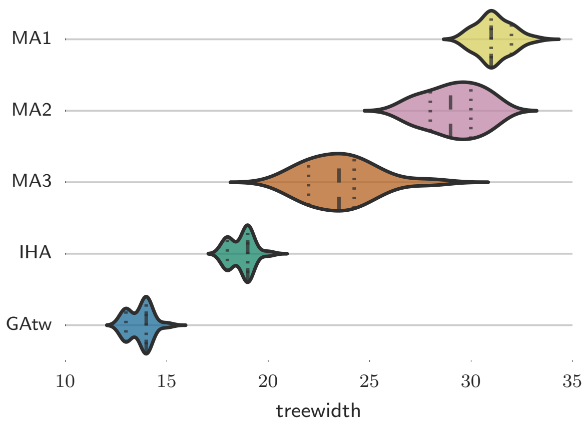


Figure 5.7: *Violin plot for instance* `fl3795`*.*

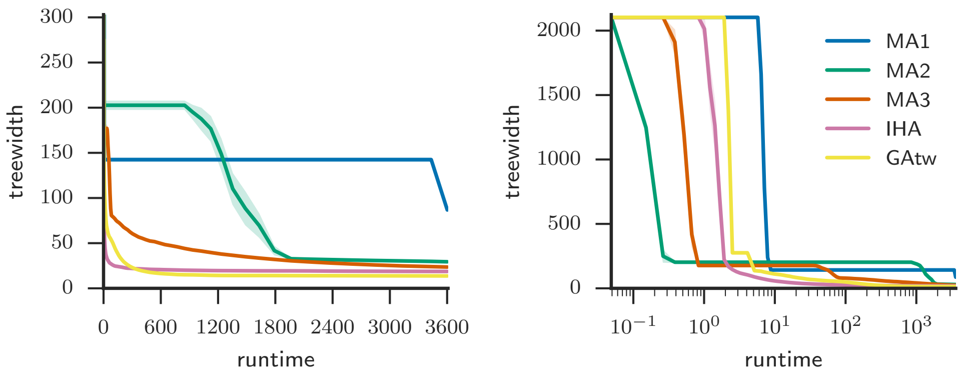


Figure 5.8: *Solution quality over time for instance* `fl3795` *(linear time scale left, logarithmic time scale right).*

Table 5.11: *Pairwise comparison of means for instance* `fl3795`*.*

		Mann-Whitney U test	
	mean diff.	$\alpha = \mathbf{10\,\%}$	$\alpha = \mathbf{5\,\%}$
IHA vs MA1	12.45	distinct	distinct
IHA vs MA2	10.50	distinct	distinct
IHA vs MA3	4.85	distinct	distinct
GA-tw vs MA1	17.45	distinct	distinct
GA-tw vs MA2	15.50	distinct	distinct
GA-tw vs MA3	9.85	distinct	distinct
MA1 vs MA2	1.95	distinct	distinct
MA1 vs MA3	7.60	distinct	distinct
MA2 vs MA3	5.65	distinct	distinct

`link`

Table 5.12: *Results for instance* `link`.

	Best		**Average**	
	treewidth	**seconds**	**treewidth**	**samples**
MA1	20	3601.6	21.6 ± 0.9	20
MA2	16	1857.7	17.9 ± 0.9	20
MA3	14	1635.1	15.5 ± 1.0	20
IHA	15	731.1	16.8 ± 1.3	20
GAtw	15	77.9	16.4 ± 1.4	20

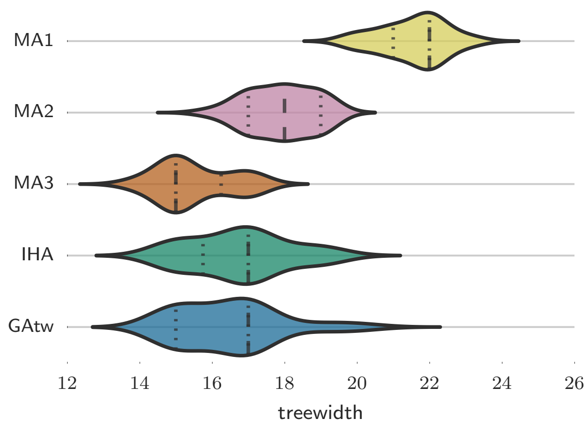


Figure 5.9: *Violin plot for instance* `link`.

While the results for all algorithms except MA1 are closely together, as Figure 5.9 illustrates, the time needed to achieve that result varies largely, as can be observed by looking at the numbers in Table 5.12 are the corresponding plots in Figure 5.10 on the next page. Interestingly, manual experiments suggest that MA1 cannot improve on this result even when given three hours of runtime.

Table 5.13 on the facing page displays the results of the tests that properly rank the results (again, no common normal distribution can be assumed). Although MA3 seems to be superior to GA-tw on this instance, the statistical test cannot reject equality of means, so in the light of statistical significance they perform equally well.

`diabetes`

Again, the most obvious feature of the results in Table 5.14 on the next page and Figure 5.11 on page 70 is the difference regarding the algorithm's time

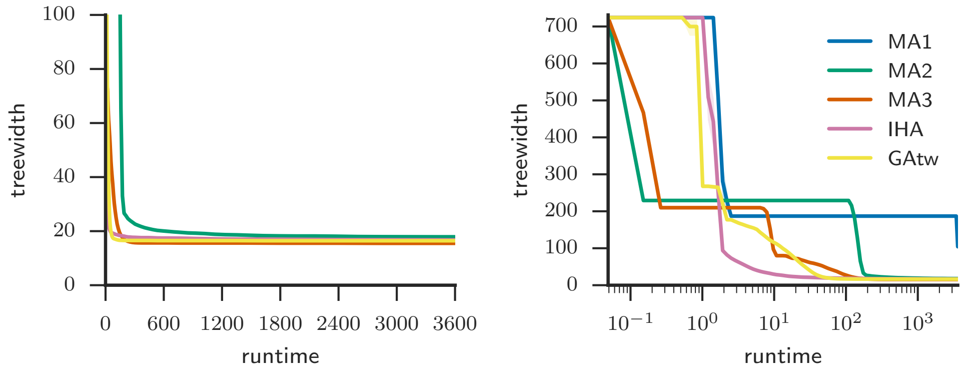


Figure 5.10: *Solution quality over time for instance* `link` *(linear time scale left, logarithmic time scale right).*

Table 5.13: *Pairwise comparison of means for instance* `link`.

		Mann-Whitney U test	
	mean diff.	$\alpha = \mathbf{10\,\%}$	$\alpha = \mathbf{5\,\%}$
IHA vs MA1	4.8	distinct	distinct
IHA vs MA2	1.15	distinct	distinct
IHA vs MA3	1.25	distinct	distinct
GA-tw vs MA1	5.1	distinct	distinct
GA-tw vs MA2	1.45	distinct	distinct
GA-tw vs MA3	0.95	equal	equal
MA1 vs MA2	3.65	distinct	distinct
MA1 vs MA3	6.05	distinct	distinct
MA2 vs MA3	2.4	distinct	distinct

Table 5.14: *Results for instance* `diabetes`.

	Best		**Average**	
	treewidth	**seconds**	**treewidth**	**samples**
MA1	9	3600.4	9.2 ± 0.4	20
MA2	8	291.0	8.3 ± 0.5	20
MA3	7	57.4	7.0 ± 0.0	20
IHA	6	2211.5	6.9 ± 0.3	20
GAtw	6	22.3	6.0 ± 0.0	20

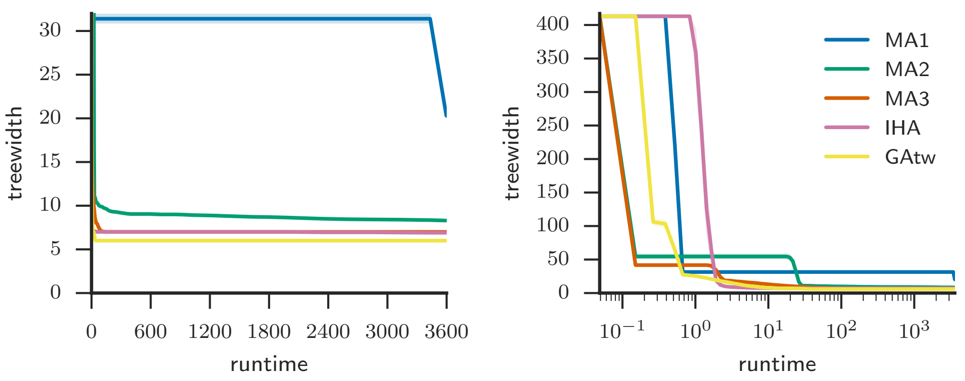


Figure 5.11: *Solution quality over time for instance* `diabetes` *(linear time scale left, logarithmic time scale right).*

responses. The data is not normally distributed; the comparison results are shown in Table 5.15.

Table 5.15: *Pairwise comparison of means for instance* `diabetes`.

		Mann-Whitney U test	
	mean diff.	$\alpha = \mathbf{10\,\%}$	$\alpha = \mathbf{5\,\%}$
IHA vs MA1	2.3	distinct	distinct
IHA vs MA2	1.4	distinct	distinct
IHA vs MA3	0.1	equal	equal
GA-tw vs MA1	3.2	distinct	distinct
GA-tw vs MA2	2.3	distinct	distinct
GA-tw vs MA3	1.0	distinct	distinct
MA1 vs MA2	0.9	distinct	distinct
MA1 vs MA3	2.2	distinct	distinct
MA2 vs MA3	1.3	distinct	distinct

1qtn

Table 5.16: *Results for instance* `1qtn`.

	Best		**Average**	
	treewidth	**seconds**	**treewidth**	**samples**
MA1	24	3508.4	24.6 ± 0.5	20
MA2	23	2589.6	24.2 ± 0.6	20
MA3	23	37.7	24.4 ± 0.7	20
IHA	23	1626.3	23.9 ± 0.3	20
GAtw	23	1052.5	23.8 ± 0.4	20

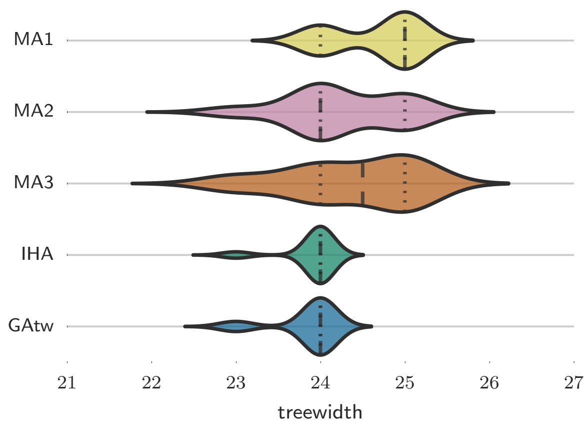


Figure 5.12: *Violin plot for instance* `1qtn`*.*

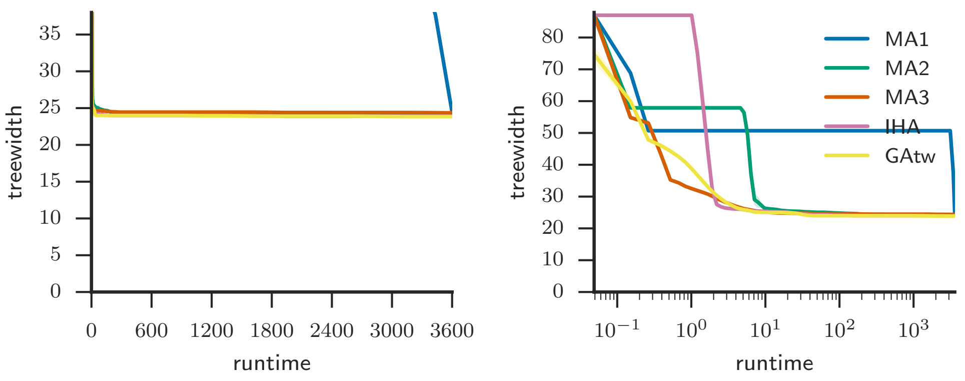


Figure 5.13: *Solution quality over time for instance* `1qtn` *(linear time scale left, logarithmic time scale right).*

Table 5.16 on the facing page and Figure 5.12 show the results to be similar. Without being able to assume normal distributions, at a significance level of 5 %, we can only differentiate between IHA and MA1, GA-tw and MA1, and GA-tw and MA3 (see Table 5.17 on the following page).

BN_16

The results in Table 5.18 on the next page and Figure 5.14 on the following page show that on this instance GA-tw achieves the best results with the lowest deviation (significance test of the non-normally distributed data in Table 5.19 on page 73). Figure 5.15 on page 73 shows a time response similar to what we see for the other instances.

Table 5.17: *Pairwise comparison of means for instance* `1qtn`.

		Mann-Whitney U test	
	mean diff.	$\alpha = \mathbf{10\,\%}$	$\alpha = \mathbf{5\,\%}$
IHA vs MA1	0.75	distinct	distinct
IHA vs MA2	0.35	equal	equal
IHA vs MA3	0.45	distinct	equal
GA-tw vs MA1	0.8	distinct	distinct
GA-tw vs MA2	0.4	distinct	equal
GA-tw vs MA3	0.5	distinct	distinct
MA1 vs MA2	0.4	equal	equal
MA1 vs MA3	0.3	equal	equal
MA2 vs MA3	0.1	equal	equal

Table 5.18: *Results for instance* `BN_16`.

	Best		**Average**	
	treewidth	**seconds**	**treewidth**	**samples**
MA1	60	3616.4	64.6 ± 3.4	20
MA2	59	3600.1	63.8 ± 2.9	20
MA3	47	2754.2	49.4 ± 1.8	20
IHA	50	1041.9	54.0 ± 3.8	20
GAtw	46	838.3	46.3 ± 0.5	20

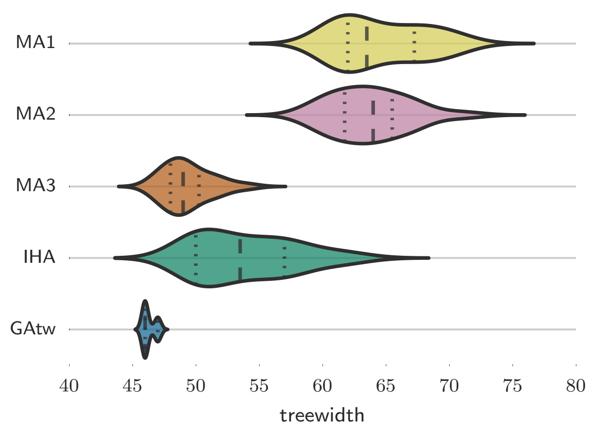


Figure 5.14: *Violin plot for instance* `BN_16`.

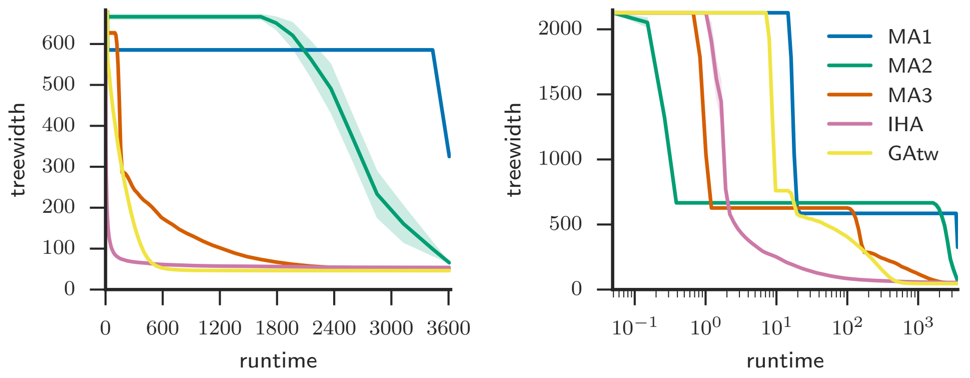


Figure 5.15: *Solution quality over time for instance* `BN_16` *(linear time scale left, logarithmic time scale right).*

Table 5.19: *Pairwise comparison of means for instance* `BN_16`.

		Mann-Whitney U test	
	mean diff.	$\alpha = \mathbf{10\,\%}$	$\alpha = \mathbf{5\,\%}$
IHA vs MA1	10.7	distinct	distinct
IHA vs MA2	9.8	distinct	distinct
IHA vs MA3	4.6	distinct	distinct
GA-tw vs MA1	18.35	distinct	distinct
GA-tw vs MA2	17.45	distinct	distinct
GA-tw vs MA3	3.05	distinct	distinct
MA1 vs MA2	0.9	equal	equal
MA1 vs MA3	15.3	distinct	distinct
MA2 vs MA3	14.4	distinct	distinct

Table 5.20: *Results for instance* `BN_20`.

	Best		**Average**	
	treewidth	**seconds**	**treewidth**	**samples**
MA1	35	3615.4	37.2 ± 1.5	20
MA2	34	3600.1	36.4 ± 1.4	20
MA3	29	3467.5	35.8 ± 3.8	20
IHA	16	2625.4	17.6 ± 0.6	20
GAtw	14	3057.9	14.4 ± 0.5	20

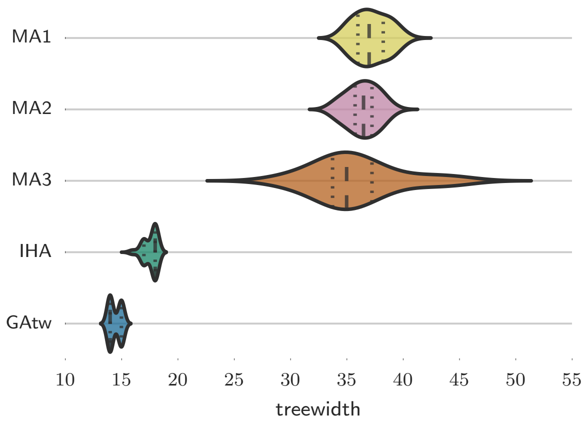


Figure 5.16: *Violin plot for instance* BN_20*.*

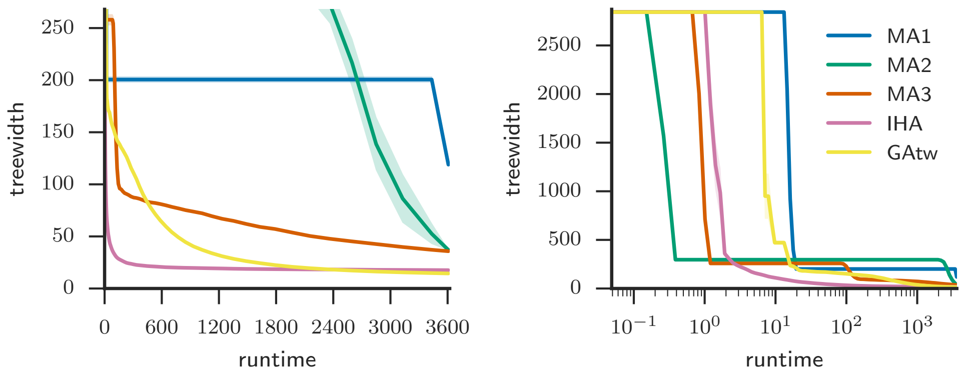


Figure 5.17: *Solution quality over time for instance* BN_20 *(linear time scale left, logarithmic time scale right).*

BN_20

Somewhat surprisingly, there is a clear difference between GA-tw and IHA and their memetic counterparts. Table 5.21 on the next page provides proof for this observation.

BN_30

With a dense field of results and different variances in Table 5.22 on the facing page and Figure 5.18 on the next page, once again we rely on statistics to clear the picture. According to the Shapiro-Wilk tests, we can assume all samples to be distributed normally ($\alpha = 5\,\%$). The variances are different, which can be seen in the violin plot (and tested using the Levene test). For Table 5.23 on page 76, we have applied the Tukey's test, the Welch's t-test (with Bonferroni correction) and the Mann-Whitney U Test (also with Bonferroni correction) to

Table 5.21: *Pairwise comparison of means for instance* `BN_20`.

		Mann-Whitney U test	
	mean diff.	$\alpha = \mathbf{10\,\%}$	$\alpha = \mathbf{5\,\%}$
IHA vs MA1	19.6	distinct	distinct
IHA vs MA2	18.85	distinct	distinct
IHA vs MA3	18.2	distinct	distinct
GA-tw vs MA1	22.75	distinct	distinct
GA-tw vs MA2	22.0	distinct	distinct
GA-tw vs MA3	21.35	distinct	distinct
MA1 vs MA2	0.75	equal	equal
MA1 vs MA3	1.4	equal	equal
MA2 vs MA3	0.65	equal	equal

Table 5.22: *Results for instance* `BN_30`.

	Best		**Average**	
	treewidth	**seconds**	**treewidth**	**samples**
MA1	72	3603.1	76.4 ± 2.6	20
MA2	64	3222.4	68.6 ± 3.1	20
MA3	62	3485.3	90.8 ± 11.7	20
IHA	56	2304.1	63.7 ± 5.3	20
GAtw	72	3388.6	74.9 ± 1.9	20

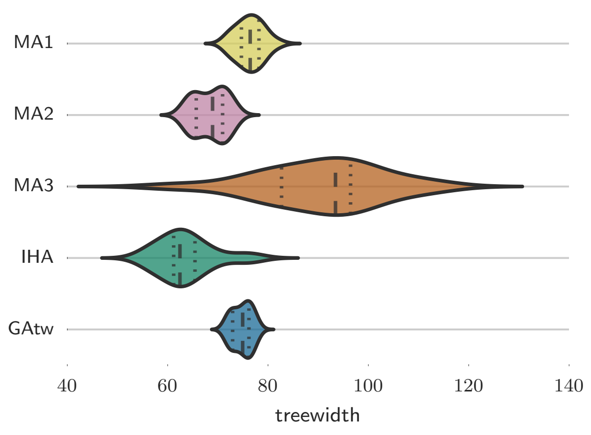


Figure 5.18: *Violin plot for instance* `BN_30`.

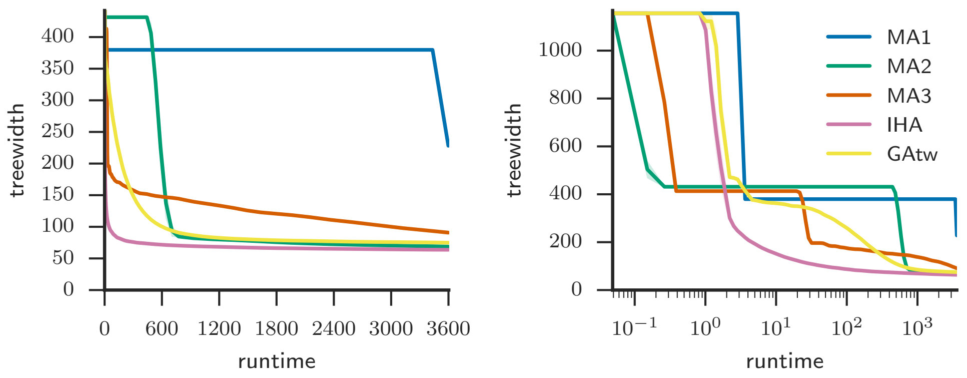


Figure 5.19: *Solution quality over time for instance* BN_30 *(linear time scale left, logarithmic time scale right).*

all algorithm pairs, at a significance level of 5 %. Although the Mann-Whitney U Test is probably more appropriate due to the heterogeneity of variance, we chose to be conservative and use the result of the Tukey's Test.

Table 5.23: *Pairwise comparison of means for instance* BN_30*, at a significance level of 5 %.*

	Diff.	**Tukey's test**	**Welch's t-test & Bonferroni**	**Mann-Whitney U & Bonferroni**
IHA vs MA1	12.75	distinct	distinct	distinct
IHA vs MA2	4.85	equal	equal	distinct
IHA vs MA3	27.1	distinct	distinct	distinct
GA-tw vs MA1	1.55	equal	equal	equal
GA-tw vs MA2	6.35	distinct	distinct	distinct
GA-tw vs MA3	15.9	distinct	distinct	distinct
MA1 vs MA2	7.9	distinct	distinct	distinct
MA1 vs MA3	14.35	distinct	distinct	distinct
MA2 vs MA3	22.25	distinct	distinct	distinct

BN_42

The results for BN_42 are given in Table 5.24 on the next page, Figure 5.20 on the facing page, and Figure 5.21 on the next page. Again, the data cannot be assumed to be distributed normally, according to the Shapiro-Wilk test. The results of the pairwise comparison using the Bonferroni-corrected Mann-Whitney U tests are shown in Table 5.25 on page 78.

BN_47

Finally, Table 5.26 on page 78, Figure 5.22 on page 78, and Figure 5.23 on page 79 present the results for BN_47. The data cannot be assumed to be

Table 5.24: *Results for instance* `BN_42`.

	Best		**Average**	
	treewidth	**seconds**	**treewidth**	**samples**
MA1	29	3601.7	32.8 ± 1.6	20
MA2	24	3264.6	26.4 ± 1.5	20
MA3	22	991.7	23.7 ± 0.9	20
IHA	23	1005.1	24.0 ± 0.8	20
GAtw	22	99.3	22.7 ± 0.9	20

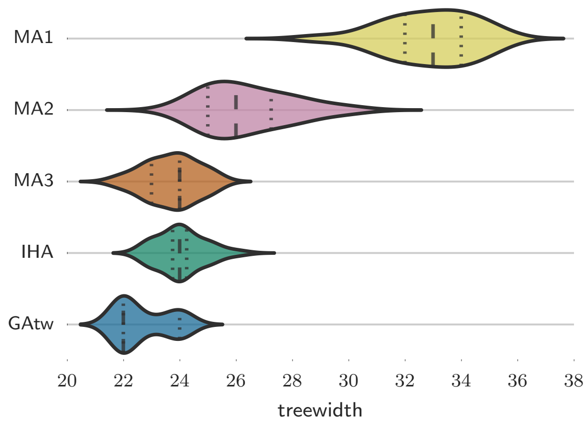


Figure 5.20: *Violin plot for instance* `BN_42`.

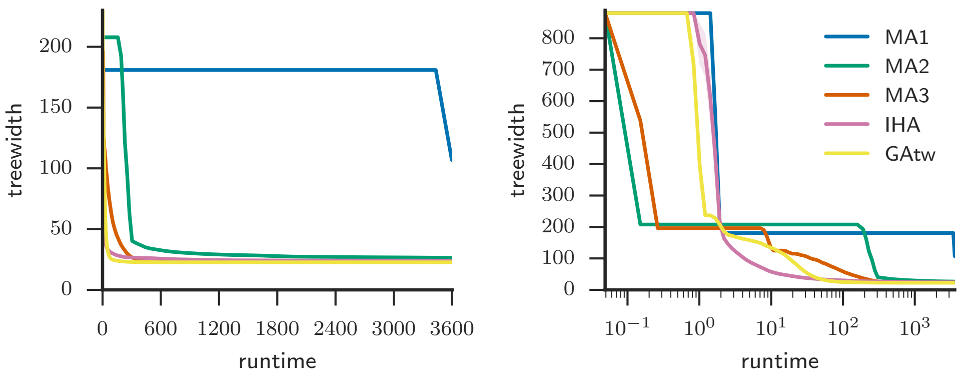


Figure 5.21: *Solution quality over time for instance* `BN_42` *(linear time scale left, logarithmic time scale right).*

Table 5.25: *Pairwise comparison of means for instance* `BN_42`.

		Mann-Whitney U test	
	mean diff.	$\alpha = \mathbf{10\,\%}$	$\alpha = \mathbf{5\,\%}$
IHA vs MA1	8.8	distinct	distinct
IHA vs MA2	2.35	distinct	distinct
IHA vs MA3	0.35	equal	equal
GA-tw vs MA1	10.15	distinct	distinct
GA-tw vs MA2	3.7	distinct	distinct
GA-tw vs MA3	1.0	distinct	distinct
MA1 vs MA2	6.45	distinct	distinct
MA1 vs MA3	9.15	distinct	distinct
MA2 vs MA3	2.7	distinct	distinct

Table 5.26: *Results for instance* `BN_47`.

	Best		**Average**	
	treewidth	**seconds**	**treewidth**	**samples**
MA1	46	3601.9	47.6 ± 0.9	20
MA2	44	779.2	44.8 ± 0.6	20
MA3	44	119.2	44.8 ± 0.9	20
IHA	44	54.8	44.4 ± 0.5	20
GAtw	44	70.5	44.8 ± 0.9	20

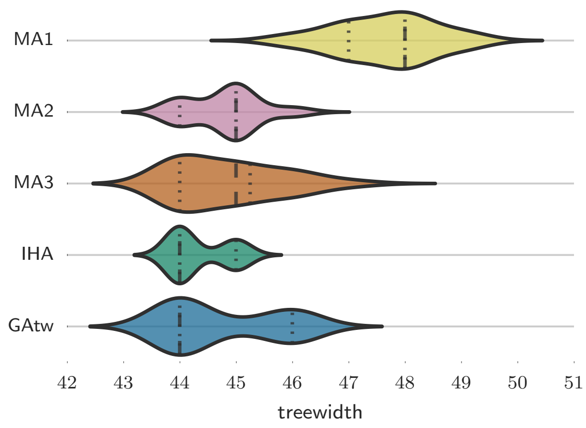


Figure 5.22: *Violin plot for instance* `BN_47`.

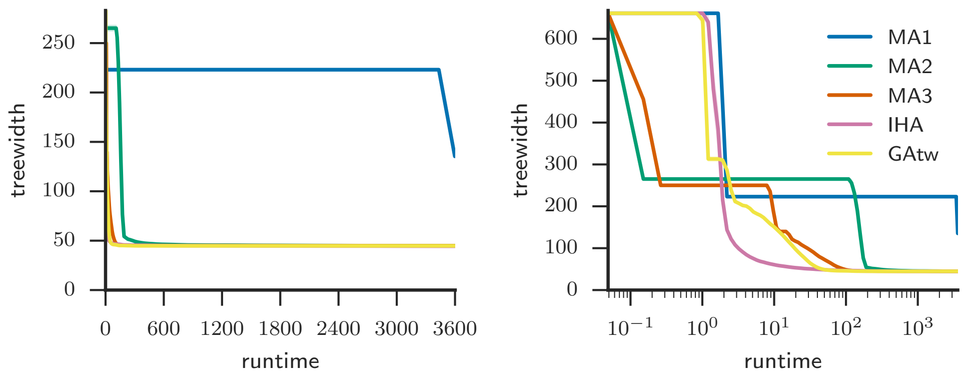


Figure 5.23: *Solution quality over time for instance* `BN_47` *(linear time scale left, logarithmic time scale right).*

distributed normally; consequently, the Mann-Whitney U test is used to judge the ranking in Table 5.27.

Table 5.27: *Pairwise comparison of means for instance* `BN_47`*.*

		Mann-Whitney U test	
	mean diff.	$\alpha = \mathbf{10\,\%}$	$\alpha = \mathbf{5\,\%}$
IHA vs MA1	3.3	distinct	distinct
IHA vs MA2	0.45	distinct	equal
IHA vs MA3	0.5	equal	equal
GA-tw vs MA1	2.9	distinct	distinct
GA-tw vs MA2	0.05	equal	equal
GA-tw vs MA3	0.1	equal	equal
MA1 vs MA2	2.85	distinct	distinct
MA1 vs MA3	2.8	distinct	distinct
MA2 vs MA3	0.05	equal	equal

5.2.2 Discussion

In the following, the algorithms are compared using the mean widths of their tree decompositions at a significance level of $\alpha = 5\,\%$. Therefore, the chance of stating that an algorithm performs better than the other when it is not, is lower than 5 %.

When comparing the memetic algorithms, we see that MA3 outperforms the other two variants. MA3 finds better results than MA2 on 33.3 % of the instances[2], and worse on 11.1 % [3]. The difference is larger for MA1, where the results are in favor of MA3 on 40.7 % of the instances[4], as opposed to the 7.4 % where MA1 takes the lead[5]. For MA1 and MA2 the difference is even clearer, for the data is in favor of MA2 for 40.7 % of the instances[6], and indifferent for the remaining instances.

MA1 is dominated by both IHA and GA-tw, where it is defeated in 59.3 % [7] and 51.2 % [8] of the instances, respectively. On a single instance, namely `knights8_8`, MA1 wins over GA-tw.

MA2 achieves equal performance on most instances, compared to IHA and GA-tw. It is outperformed in 48.1 % of the instances in the case of IHA [9]. Compared to GA-tw, MA2 is defeated 40.7 % of the time[10], but manages to find better solutions for 11.1 % of the instances[11].

Finally, MA3's performance is equal to IHA's on the majority of the instances (66.6 %). It finds better results than IHA on 7.4 % of the instances[12], while IHA performs better on 25.9 % [13]. The performance for MA3 versus GA-tw is slightly less impressive: 7.4 % in favor of MA3 [14], and 40.7 % in favor of GA-tw[15]. Still, on more than half of the instances the algorithms are on par.

The result is presented graphically in Figure 5.24 on the next page.

[2]rl5934, pcb3038, fl3795, link, diabetes, BN_16, BN_42, 1or7, 1on2 [3]fnl4461, BN_30, 1a62 [4]rl5934, pcb3038, fl3795, link, diabetes, BN_16, BN_42, BN_47, 1a62, 1or7, 1on2 [5]fnl4461, BN_30 [6]rl5934, pcb3038, fl3795, link, diabetes, BN_30, BN_42, BN_47, 1a62, 1or7, 1on2 [7]rl5934, pcb3038, fnl4461, fl3795, link, diabetes, 1qtn, BN_16, BN_20, BN_30, BN_42, BN_47, knights8_8, 1a62, 1or7, 1on2 [8]rl5934, pcb3038, fnl4461, fl3795, link, diabetes, 1qtn, BN_16, BN_20, BN_42, BN_47, 1a62, 1or7, 1on2 [9]rl5934, pcb3038, fnl4461, fl3795, link, diabetes, BN_16, BN_20, BN_42, knights8_8, 1or7, 1on2, BN_0 [10]rl5934, pcb3038, fnl4461, fl3795, link, diabetes, 1qtn, BN_16, BN_20, BN_42, 1on2 [11]BN_30, knights8_8, 1or7 [12]link, BN_16 [13]fnl4461, fl3795, 1qtn, BN_20, BN_30, 1a62, BN_0 [14]knights8_8, 1or7 [15]rl5934, pcb3038, fnl4461, fl3795, diabetes, 1qtn, BN_16, BN_20, BN_30, BN_42, 1a62

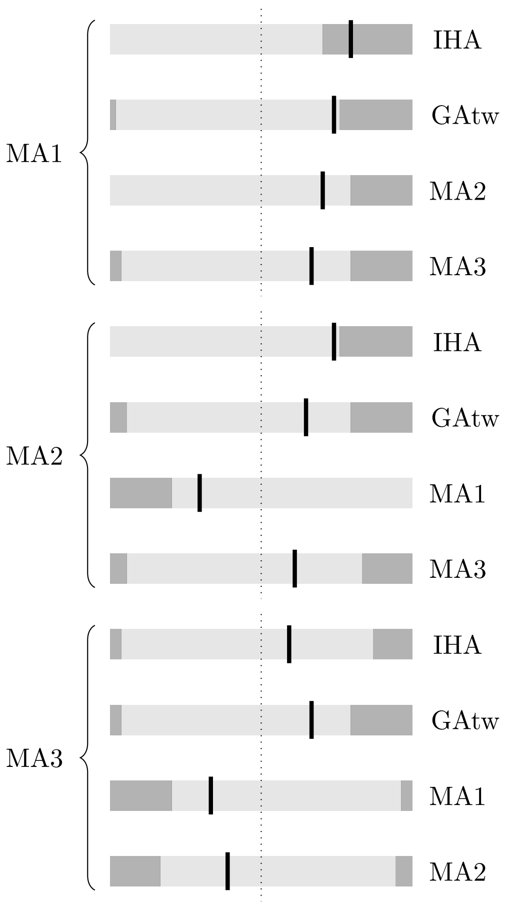


Figure 5.24: *Result of the comparisons on the validation instances. Dark areas represent instances where the respective side performs significantly better than the other (significance level $\alpha = 5\,\%$). The remaining light areas in between represent instances where no performance difference could be identified. The verical black bars signify the result of the respective comparison.*

Chapter 6

Conclusions

In the course of this work, we have designed three different memetic algorithms. Their performance has been evaluated by comparing them to each other, as well as to the state-of-the-art algorithms BB-tw [6], QuickBB [21], TabuTW [16], ACS+ILS [23, 24], GA-tw [53, 46], and IHA [45].

Our algorithms have been optimized by parameter tuning software. To ensure that the results represent real-world performance, we have used separate sets of instances for training/tuning (79 instances) and for validation (27 instances). In order for the results to be meaningful, statistical significance tests have been employed for ranking the algorithms.

The proposed memetic algorithms differ with respect to solution quality, i.e., the width of the resulting tree decomposition, where MA3 takes the lead in almost all instances. They also differ substantially regarding their time response, with MA3 being quicker than MA2, which is in turn quicker than MA1. So overall, MA3 turned out to be superior to the more complex variants MA1 and MA2.

In general, our memetic algorithms are competitive, but they do not dominate all state-of-the-art solvers for this problem. Nevertheless, MA3 has been able to find new best known upper bounds on 8 benchmark instances of the Second DIMACS Implementation Challenge. One of them is also found by MA2.

The results suggest that memetic algorithms are promising and worth investigating. Future work includes experiments using parallel implementations of MAs, where a natural distinction between the phases of a memetic algorithm can help to optimize for concurrency. Since the way of combining two heuristics into a MA has such a large impact on the performance, it might be worth investigating more variants, while aiming at an understanding of the underlying principles.

Appendix A

Performance on Validation Instances, continued

This Chapter lists those instances that have not been mentioned in Section 5.2.1 on page 61, on which the tested algorithms perform very similarly.

Table A.1: *Results for instance* `barley`.

	Best		**Average**	
	treewidth	**seconds**	**treewidth**	**samples**
MA1	7	574.1	7.0 ± 0.0	20
MA2	7	0.8	7.0 ± 0.0	20
MA3	7	0.1	7.0 ± 0.0	20
IHA	7	1.1	7.0 ± 0.0	20
GAtw	7	0.1	7.0 ± 0.0	20

Table A.2: *Results for instance* `pathfinder`.

	Best		**Average**	
	treewidth	**seconds**	**treewidth**	**samples**
MA1	6	1051.7	6.0 ± 0.0	20
MA2	6	1.4	6.0 ± 0.0	20
MA3	6	0.2	6.0 ± 0.0	20
IHA	6	1.1	6.0 ± 0.0	20
GAtw	6	0.3	6.0 ± 0.0	20

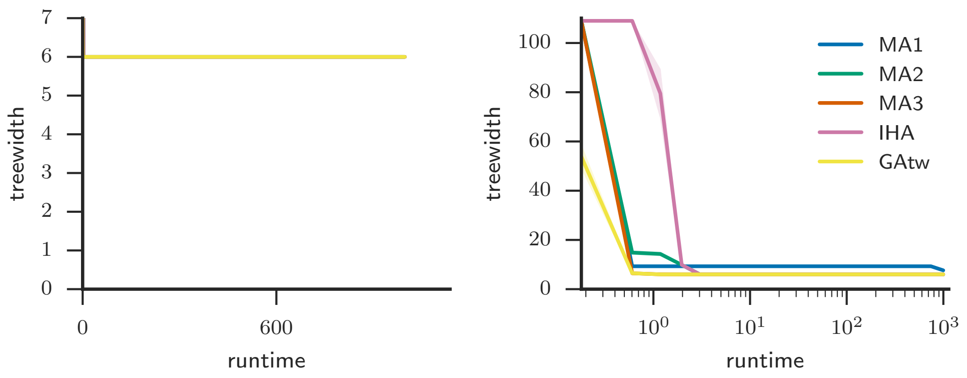


Figure A.1: *Solution quality over time for instance* `pathfinder` *(linear time scale left, logarithmic time scale right).*

Table A.3: *Results for instance* `oesoca+`*.*

	Best		**Average**	
	treewidth	**seconds**	**treewidth**	**samples**
MA1	11	723.8	11.0 ± 0.0	20
MA2	11	1.1	11.0 ± 0.0	20
MA3	11	0.2	11.0 ± 0.0	20
IHA	11	1.1	11.0 ± 0.0	20
GAtw	11	0.3	11.0 ± 0.0	20

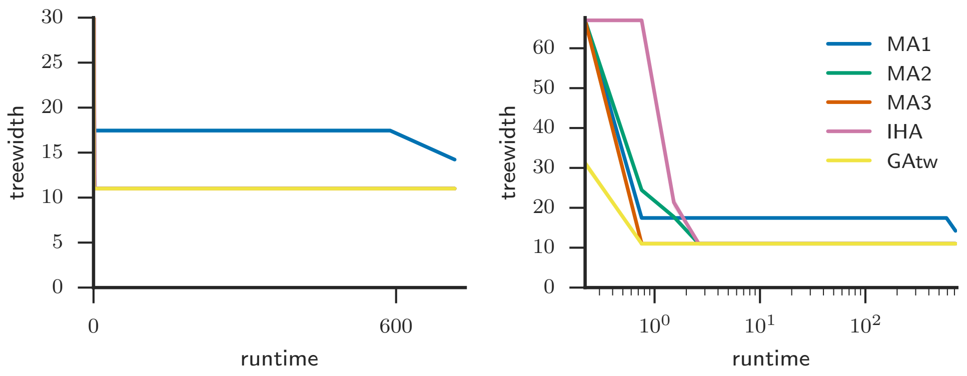


Figure A.2: *Solution quality over time for instance* `oesoca+` *(linear time scale left, logarithmic time scale right).*

Table A.4: *Results for instance* `pigs`.

	Best		**Average**	
	treewidth	**seconds**	**treewidth**	**samples**
MA1	10	3600.5	10.0 ± 0.2	20
MA2	10	24.1	10.0 ± 0.0	20
MA3	9	978.9	10.1 ± 0.4	20
IHA	9	2173.2	10.0 ± 0.2	20
GAtw	10	6.9	10.1 ± 0.3	20

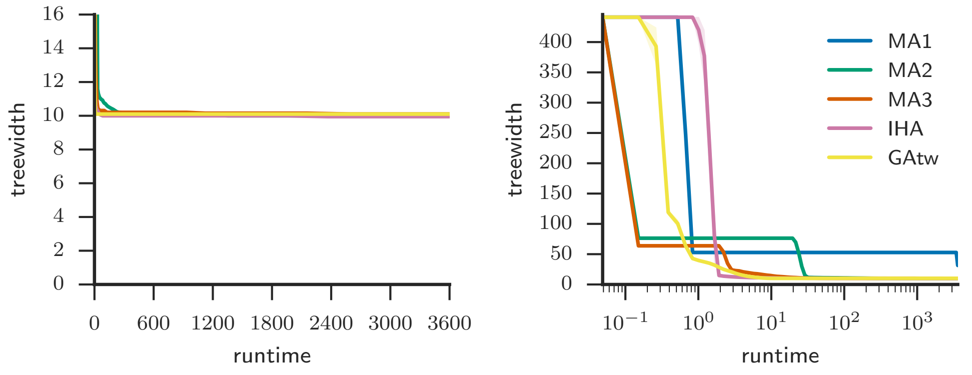


Figure A.3: *Solution quality over time for instance* `pigs` *(linear time scale left, logarithmic time scale right).*

Table A.5: *Pairwise comparison of means for instance* `pigs`.

		Mann-Whitney U test	
	mean diff.	$\alpha =$ **10 %**	$\alpha =$ **5 %**
IHA vs MA1	0.1	equal	equal
IHA vs MA2	0.05	equal	equal
IHA vs MA3	0.15	equal	equal
GA-tw vs MA1	0.05	equal	equal
GA-tw vs MA2	0.1	equal	equal
GA-tw vs MA3	0.0	equal	equal
MA1 vs MA2	0.05	equal	equal
MA1 vs MA3	0.05	equal	equal
MA2 vs MA3	0.1	equal	equal

Table A.6: *Results for instance* `water`.

	Best		**Average**	
	treewidth	**seconds**	**treewidth**	**samples**
MA1	9	353.5	9.0 ± 0.0	20
MA2	9	0.6	9.0 ± 0.0	20
MA3	9	0.2	9.0 ± 0.0	20
IHA	9	1.1	9.0 ± 0.0	20
GAtw	9	0.2	9.0 ± 0.0	20

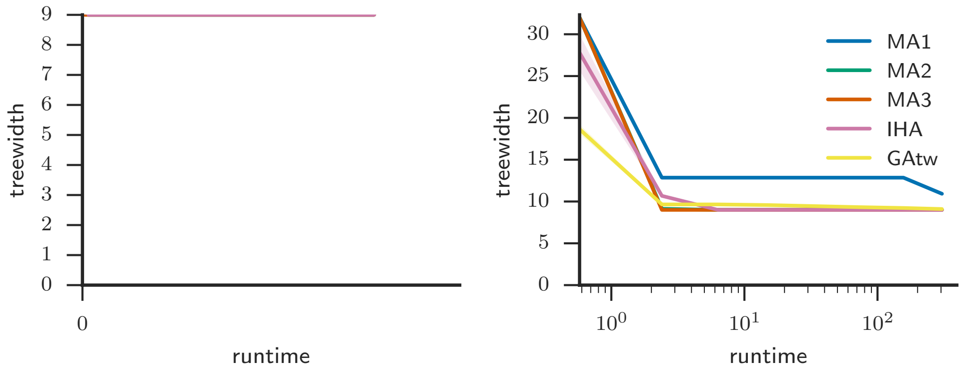


Figure A.4: *Solution quality over time for instance* `water` *(linear time scale left, logarithmic time scale right).*

Table A.7: *Results for instance* `munin1`.

	Best		**Average**	
	treewidth	**seconds**	**treewidth**	**samples**
MA1	11	3600.1	11.0 ± 0.0	20
MA2	11	8.1	11.0 ± 0.0	20
MA3	11	3.3	11.0 ± 0.0	20
IHA	11	1.3	11.0 ± 0.0	20
GAtw	11	1.9	11.0 ± 0.0	20

Table A.8: *Results for instance* `1ubq`.

	Best		**Average**	
	treewidth	**seconds**	**treewidth**	**samples**
MA1	12	1520.5	12.0 ± 0.0	20
MA2	12	2.3	12.0 ± 0.0	20
MA3	12	0.8	12.0 ± 0.0	20
IHA	12	1.2	12.0 ± 0.0	20
GAtw	12	0.7	12.0 ± 0.0	20

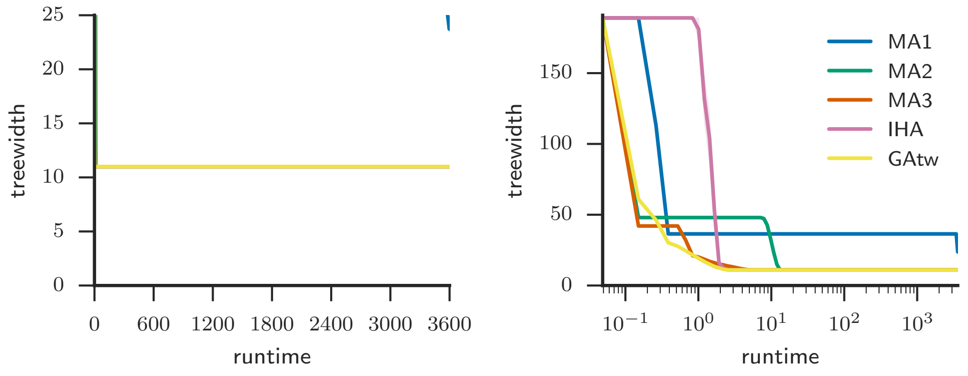


Figure A.5: *Solution quality over time for instance* `munin1` *(linear time scale left, logarithmic time scale right).*

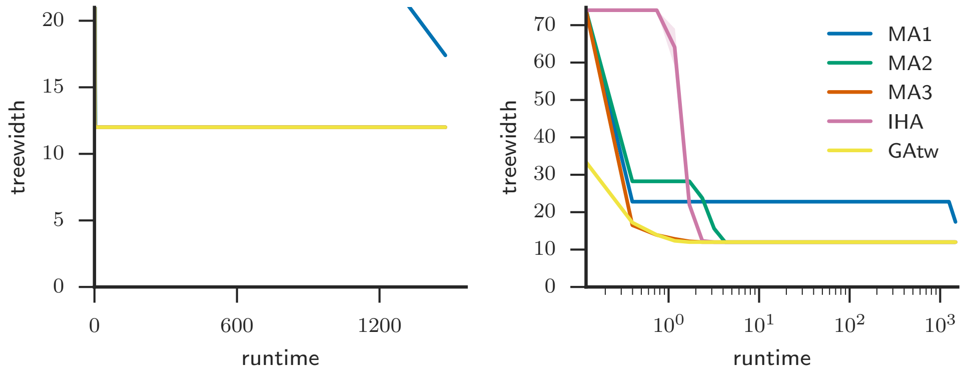


Figure A.6: *Solution quality over time for instance* `1ubq` *(linear time scale left, logarithmic time scale right).*

Table A.9: *Results for instance* `knights8_8`.

	Best		**Average**	
	treewidth	**seconds**	**treewidth**	**samples**
MA1	16	1238.1	16.4 ± 0.5	20
MA2	16	89.3	16.3 ± 0.5	20
MA3	16	3.7	16.2 ± 0.4	20
IHA	16	11.7	16.0 ± 0.0	20
GAtw	16	3.2	18.6 ± 1.3	20

Table A.10: *Pairwise comparison of means for instance* `knights8_8`.

	mean diff.	**Mann-Whitney U test** $\alpha = 10\,\%$	**Mann-Whitney U test** $\alpha = 5\,\%$
IHA vs MA1	0.45	distinct	distinct
IHA vs MA2	0.3	distinct	distinct
IHA vs MA3	0.15	equal	equal
GA-tw vs MA1	2.2	distinct	distinct
GA-tw vs MA2	2.35	distinct	distinct
GA-tw vs MA3	2.5	distinct	distinct
MA1 vs MA2	0.15	equal	equal
MA1 vs MA3	0.3	equal	equal
MA2 vs MA3	0.15	equal	equal

Table A.11: *Results for instance* `1a62`.

	Best		**Average**	
	treewidth	**seconds**	**treewidth**	**samples**
MA1	37	3600.1	37.6 ± 0.5	20
MA2	36	160.8	36.2 ± 0.4	20
MA3	36	23.3	36.8 ± 0.8	20
IHA	36	6.0	36.0 ± 0.0	20
GAtw	36	88.6	36.0 ± 0.0	20

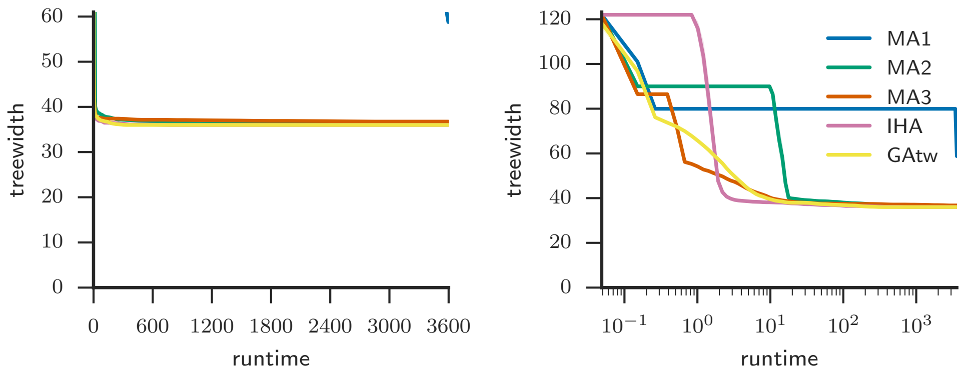


Figure A.7: *Solution quality over time for instance* `1a62` *(linear time scale left, logarithmic time scale right).*

Table A.12: *Pairwise comparison of means for instance* `1a62`.

	mean diff.	**Mann-Whitney U test** $\alpha = \mathbf{10\,\%}$	$\alpha = \mathbf{5\,\%}$
IHA vs MA1	1.6	distinct	distinct
IHA vs MA2	0.15	equal	equal
IHA vs MA3	0.75	distinct	distinct
GA-tw vs MA1	1.6	distinct	distinct
GA-tw vs MA2	0.15	equal	equal
GA-tw vs MA3	0.75	distinct	distinct
MA1 vs MA2	1.45	distinct	distinct
MA1 vs MA3	0.85	distinct	distinct
MA2 vs MA3	0.6	distinct	distinct

Table A.13: *Results for instance* `1sem`.

	Best treewidth	seconds	**Average** treewidth	**samples**
MA1	26	1237.5	26.0 ± 0.0	20
MA2	26	1.9	26.0 ± 0.0	20
MA3	26	1.4	26.0 ± 0.0	20
IHA	26	1.2	26.0 ± 0.0	20
GAtw	26	1.4	26.0 ± 0.0	20

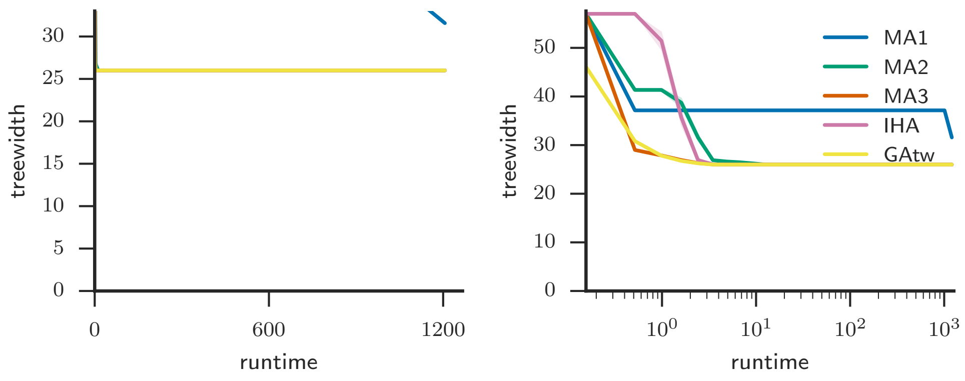


Figure A.8: *Solution quality over time for instance* `1sem` *(linear time scale left, logarithmic time scale right).*

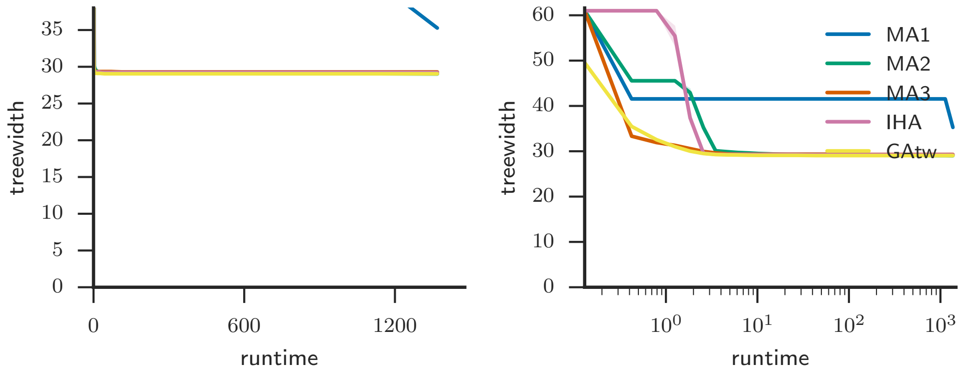


Figure A.9: *Solution quality over time for instance* `1pwt` *(linear time scale left, logarithmic time scale right).*

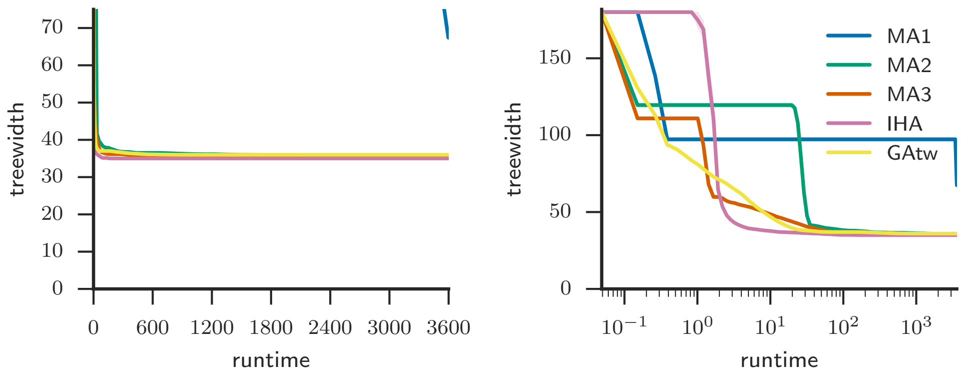


Figure A.10: *Solution quality over time for instance* `1or7` *(linear time scale left, logarithmic time scale right).*

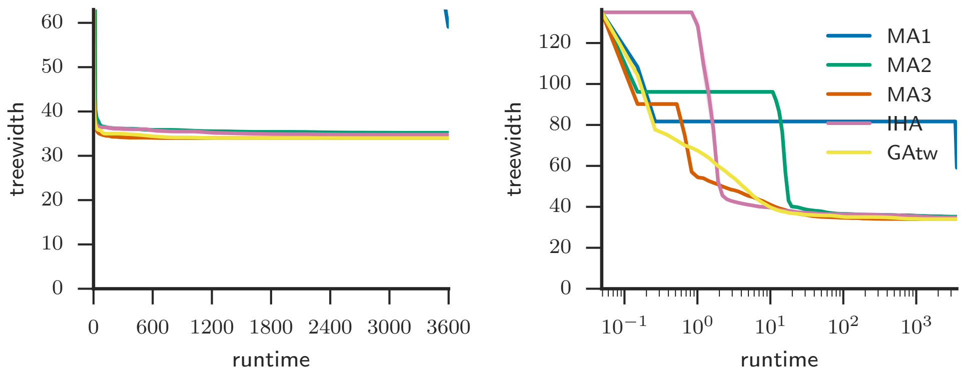


Figure A.11: *Solution quality over time for instance* `1on2` *(linear time scale left, logarithmic time scale right).*

Table A.14: *Results for instance* `1pwt`.

	Best		Average	
	treewidth	**seconds**	**treewidth**	**samples**
MA1	29	1326.8	29.0 ± 0.0	20
MA2	29	2.7	29.0 ± 0.0	20
MA3	29	1.8	29.3 ± 0.5	20
IHA	29	1.3	29.2 ± 0.4	20
GAtw	29	1.8	29.0 ± 0.2	20

Table A.15: *Results for instance* `1or7`.

	Best		Average	
	treewidth	**seconds**	**treewidth**	**samples**
MA1	36	3600.2	37.6 ± 0.6	20
MA2	35	401.6	35.6 ± 0.7	20
MA3	35	65.1	35.1 ± 0.4	20
IHA	35	17.4	35.0 ± 0.0	20
GAtw	36	176.6	36.0 ± 0.0	20

Table A.16: *Pairwise comparison of means for instance* `1or7`.

		Mann-Whitney U test	
	mean diff.	$\alpha = \mathbf{10\,\%}$	$\alpha = \mathbf{5\,\%}$
IHA vs MA1	2.55	distinct	distinct
IHA vs MA2	0.6	distinct	distinct
IHA vs MA3	0.1	equal	equal
GA-tw vs MA1	1.55	distinct	distinct
GA-tw vs MA2	0.4	distinct	distinct
GA-tw vs MA3	0.9	distinct	distinct
MA1 vs MA2	1.95	distinct	distinct
MA1 vs MA3	2.45	distinct	distinct
MA2 vs MA3	0.5	distinct	distinct

Table A.17: *Results for instance* `1on2`.

	Best		Average	
	treewidth	**seconds**	**treewidth**	**samples**
MA1	36	3600.1	36.5 ± 0.5	20
MA2	34	2124.9	35.2 ± 0.6	20
MA3	34	50.8	34.0 ± 0.0	20
IHA	34	179.2	34.8 ± 1.8	20
GAtw	34	281.2	34.0 ± 0.0	20

Table A.18: *Pairwise comparison of means for instance* `1on2`*.*

		Mann-Whitney U test	
	mean diff.	$\alpha = \mathbf{10\,\%}$	$\alpha = \mathbf{5\,\%}$
IHA vs MA1	1.7	distinct	distinct
IHA vs MA2	0.4	distinct	distinct
IHA vs MA3	0.8	equal	equal
GA-tw vs MA1	2.5	distinct	distinct
GA-tw vs MA2	1.2	distinct	distinct
GA-tw vs MA3	0.0	(equal)	(equal)
MA1 vs MA2	1.3	distinct	distinct
MA1 vs MA3	2.5	distinct	distinct
MA2 vs MA3	1.2	distinct	distinct

Table A.19: *Results for instance* `1oai`*.*

	Best		**Average**	
	treewidth	**seconds**	**treewidth**	**samples**
MA1	22	1414.2	22.0 ± 0.0	20
MA2	22	4.8	22.0 ± 0.0	20
MA3	22	2.6	22.0 ± 0.2	20
IHA	22	2.5	22.0 ± 0.0	20
GAtw	22	4.2	22.0 ± 0.0	20

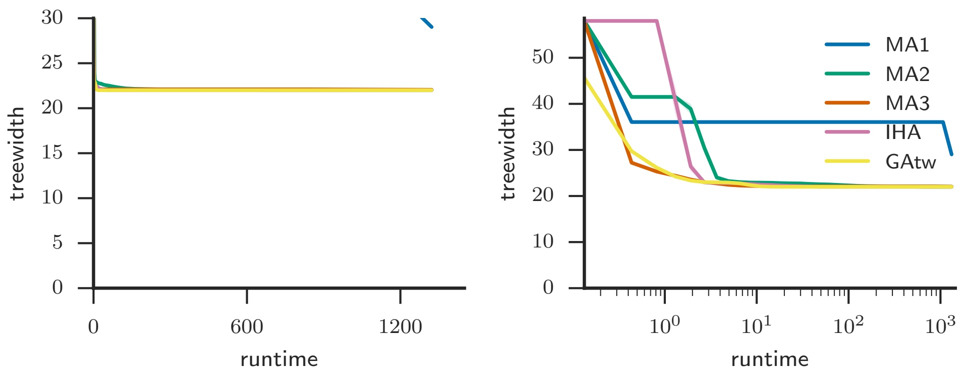


Figure A.12: *Solution quality over time for instance* `1oai` *(linear time scale left, logarithmic time scale right).*

Table A.20: *Results for instance* `BN_0`.

	Best		**Average**	
	treewidth	**seconds**	**treewidth**	**samples**
MA1	21	3600.1	22.0 ± 0.2	20
MA2	22	4.4	22.0 ± 0.0	20
MA3	22	5.5	22.0 ± 0.0	20
IHA	21	365.2	21.6 ± 0.5	20
GAtw	22	2.7	22.4 ± 0.8	20

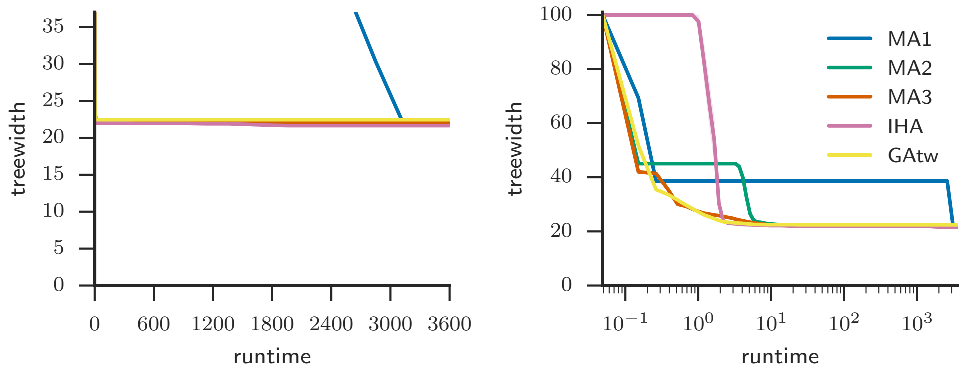


Figure A.13: *Solution quality over time for instance* `BN_0` *(linear time scale left, logarithmic time scale right).*

Table A.21: *Pairwise comparison of means for instance* `BN_0`.

		Mann-Whitney U test	
	mean diff.	$\alpha = \mathbf{10\,\%}$	$\alpha = \mathbf{5\,\%}$
IHA vs MA1	1.7	distinct	equal
IHA vs MA2	0.4	distinct	distinct
IHA vs MA3	0.8	distinct	distinct
GA-tw vs MA1	2.5	distinct	equal
GA-tw vs MA2	1.2	distinct	equal
GA-tw vs MA3	0.0	distinct	equal
MA1 vs MA2	1.3	equal	equal
MA1 vs MA3	2.5	equal	equal
MA2 vs MA3	1.2	(equal)	(equal)

Glossary

ACS+ILS ant colony system in combination with iterated local search for treewidth. 56–58, 83
ANOVA analysis of variance. 61, 62

BB-tw branch-and-bound algorithm for treewidth by Bachoore and Bodlaender. 56–58, 83

Competition in MA1: selection mechanism (as in GA) that may replace one of two involved individuals. 18
Cooperation in MA1: recombination of two individuals using a crossover operator. 16
CSP constraint satisfaction problem. 1, 2, 9

DIMACS Center for Discrete Mathematics & Theoretical Computer Science. See `http://dimacs.rutgers.edu`. ix, xi, 6, 29, 55, 83

GA genetic algorithm. 11–13, 15, 16, 18, 23, 25, 34, 35, 97
GA-tw Genetic Algorithm for treewidth. 25, 35, 56–58, 63, 64, 66–76, 78–80, 83, 87, 90, 91, 93–95, 97

IHA Iterative Heuristic Algorithm for treewidth. 25, 35, 56–58, 61, 63, 64, 66, 67, 69–76, 78–80, 83, 87, 90, 91, 93–95, 97
ILS iterated local search. 11, 15, 20, 24, 25, 34, 35
IM insertion mutation. 23, 25

LOWESS locally weighted scatterplot smoothing. 35
LS localsearch. 12

MA memetic algorithm. 12, 13, 23, 83, 97
MA1 MA based on [43]. 6, 15, 16, 19, 23, 25, 30–34, 36–42, 56–58, 63–73, 75, 76, 78–80, 83, 87, 90, 91, 93–95, 97, 98
MA2 MA designed along the lines of [59, 60]. 6, 13, 16, 19, 23, 25, 30–33, 35, 43–49, 56–58, 63–67, 69, 70, 72, 73, 75, 76, 78–80, 83, 87, 90, 91, 93–95
MA3 MA formed by combining GA-tw and IHA. 6, 16, 25, 30–33, 35, 50–53, 56–58, 61, 63, 64, 66–73, 75, 76, 78–80, 83, 87, 90, 91, 93–95
Meme 16

NP nondeterministic-polynomial-time complexity class in complexity theory.

Opponent in MA1: individual that is available for competition. 16

P polynomial-time complexity class in complexity theory.
Partner in MA1: individual that is available for cooperation. 16
POS-crossover position-based crossover. 16, 19, 22, 23, 25, 27

QuickBB branch-and-bound algorithm for treewidth by Gogate and Dechter. 56–58, 83

SMAC Sequential Model-based Algorithm Configuration. 29, 34, 43

TabuTW tabu search for treewidth. 56–58, 83
Treewidth 9

Bibliography

[1] A. B. Adcock, B. D. Sullivan, and M. W. Mahoney, “Tree decompositions and social graphs,” *arXiv preprint arXiv:1411.1546*, 2014 (cit. on p. 9).

[2] J. Alber, F. Dorn, and R. Niedermeier, “Experimental evaluation of a tree decomposition-based algorithm for vertex cover on planar graphs,” *Discrete Applied Mathematics*, vol. 145, no. 2, pp. 219–231, 2005. DOI: 10.1016/j.dam.2004.01.013 (cit. on p. 9).

[3] E. Amir, “Efficient approximation for triangulation of minimum treewidth,” in *Proceedings of the Seventeenth Conference on Uncertainty in Artificial Intelligence*, ser. UAI’01, Seattle, Washington: Morgan Kaufmann Publishers Inc., 2001, pp. 7–15 (cit. on p. 10).

[4] S. Andreassen, R. Hovorka, J. Benn, K. G. Olesen, and E. R. Carson, “A model-based approach to insulin adjustment,” in *Proceedings of the Third Conference on Artificial Intelligence in Medicine*, M. Stefanelli, A. Hasman, M. Fieschi, and J. Talmon, Eds., Springer-Verlag, 1991, pp. 239–248 (cit. on p. 59).

[5] S. Arnborg, D. G. Corneil, and A. Proskurowski, “Complexity of finding embeddings in a k-tree,” *SIAM Journal on Algebraic Discrete Methods*, vol. 8, no. 2, pp. 277–284, 1987 (cit. on p. 6).

[6] E. H. Bachoore and H. L. Bodlaender, “A branch and bound algorithm for exact, upper, and lower bounds on treewidth,” in *Algorithmic Aspects in Information and Management*, ser. Lecture Notes in Computer Science, S.-W. Cheng and C. Poon, Eds., vol. 4041, Springer Berlin Heidelberg, 2006, pp. 255–266. DOI: 10.1007/11775096_24 (cit. on pp. 10, 55, 83, 97).

[7] A. Becker and D. Geiger, “A sufficiently fast algorithm for finding close to optimal clique trees,” *Artificial Intelligence*, vol. 125, no. 1–2, pp. 3–17, 2001. DOI: 10.1016/S0004-3702(00)00075-8 (cit. on p. 10).

[8] C. Blum, M. J. B. Aguilera, A. Roli, and M. Sampels, Eds., *Hybrid Metaheuristics, An Emerging Approach to Optimization*, vol. 114, ser. Studies in Computational Intelligence, Springer, 2008 (cit. on p. 13).

[9] H. L. Bodlaender, B. M. P. Jansen, and S. Kratsch, “Preprocessing for treewidth: a combinatorial analysis through kernelization,” *SIAM Journal on Discrete Mathematics*, vol. 27, no. 4, pp. 2108–2142, Dec. 17, 2013 (cit. on p. 12).

[10] H. L. Bodlaender and A. M. C. A. Koster, “Treewidth computations I. upper bounds,” *Information and Computation*, vol. 208, no. 3, pp. 259–275, 2010. DOI: 10.1016/j.ic.2009.03.008 (cit. on p. 9).

[11] B. Bontoux, C. Artigues, and D. Feillet, “A memetic algorithm with a large neighborhood crossover operator for the generalized traveling salesman problem,” *Computers & Operations Research*, vol. 37, no. 11, pp. 1844–1852, 2010. DOI: 10.1016/j.cor.2009.05.004 (cit. on p. 13).

[12] V. Bouchitté, D. Kratsch, H. Müller, and I. Todinca, “On treewidth approximations,” *Discrete Applied Mathematics*, vol. 136, no. 2–3, pp. 183–196, 2004, The 1st Cologne-Twente Workshop on Graphs and Combinatorial Optimization. DOI: 10.1016/S0166-218X(03)00440-2 (cit. on p. 10).

[13] J.-W. van den Broek and H. Bodlaender. (Dec. 1, 2014). Treewidthlib, [Online]. Available: http://www.cs.uu.nl/research/projects/treewidthlib (cit. on pp. 58–60).

[14] Center for Discrete Mathematics and Theoretical Computer Science. (Dec. 1, 2014). DIMACS implementation challenges, [Online]. Available: http://dimacs.rutgers.edu/Challenges (cit. on pp. 6, 29).

[15] ——, (Dec. 1, 2014). Graph-coloring instances, [Online]. Available: http://mat.gsia.cmu.edu/COLOR/instances.html (cit. on p. 29).

[16] F. Clautiaux, A. Moukrim, S. Nègre, and J. Carlier, “Heuristic and metaheuristic methods for computing graph treewidth,” *RAIRO - Operations Research*, vol. 38, pp. 13–26, 01 Jan. 2004. DOI: 10.1051/ro:2004011 (cit. on pp. 11, 55, 83).

[17] W. S. Cleveland, “Robust locally weighted regression and smoothing scatterplots,” *Journal of the American Statistical Association*, vol. 74, no. 368, pp. 829–836, 1979 (cit. on p. 35).

[18] R. Dawkins, *The Selfish Gene*. Oxford, UK: Oxford University Press, 1976 (cit. on pp. 12, 16).

[19] F. V. Fomin, D. Kratsch, and I. Todinca, “Exact (exponential) algorithms for treewidth and minimum fill-in,” in *Automata, Languages and Programming*, ser. Lecture Notes in Computer Science, J. Díaz, J. Karhumäki, A. Lepistö, and D. Sannella, Eds., vol. 3142, Springer Berlin Heidelberg, 2004, pp. 568–580. DOI: 10.1007/978-3-540-27836-8_49 (cit. on p. 10).

[20] P. Galinier and J.-K. Hao, "Hybrid evolutionary algorithms for graph coloring," *Journal of Combinatorial Optimization*, vol. 3, no. 4, pp. 379–397, 1999. DOI: 10.1023/A:1009823419804 (cit. on pp. 13, 23).

[21] V. Gogate and R. Dechter, "A complete anytime algorithm for treewidth," in *Proceedings of the 20th Conference on Uncertainty in Artificial Intelligence*, AUAI Press, 2004, pp. 201–208 (cit. on pp. 10, 55, 83, 98).

[22] G. Gottlob, N. Leone, and F. Scarcello, "A comparison of structural CSP decomposition methods," *Artificial Intelligence*, vol. 124, no. 2, pp. 243–282, 2000. DOI: 10.1016/S0004-3702(00)00078-3 (cit. on p. 2).

[23] T. Hammerl, "Ant colony optimization for tree and hypertree decompositions," Master's thesis, Vienna University of Technology, 2009 (cit. on pp. 11, 15, 55, 83).

[24] T. Hammerl and N. Musliu, "Ant colony optimization for tree decompositions," in *Evolutionary Computation in Combinatorial Optimization, 10th European Conference, EvoCOP 2010, Istanbul, Turkey*, P. I. Cowling and P. Merz, Eds., Springer, 2010, pp. 95–106. DOI: 10.1007/978-3-642-12139-5_9 (cit. on pp. 11, 15, 55, 83).

[25] T. Hammerl, N. Musliu, and W. Schafhauser, "Metaheuristic algorithms and tree decomposition," in *Handbook of Computational Intelligence*. to appear (cit. on p. 11).

[26] M. Held and R. M. Karp, "A dynamic programming approach to sequencing problems," *Journal of the Society for Industrial & Applied Mathematics*, vol. 10, no. 1, pp. 196–210, 1962 (cit. on p. 10).

[27] A. Hildebrandt and M. Krupp, "Algorithms for the maximum weight connected k-induced subgraph problem," *Combinatorial Optimization and Applications*, p. 268, (cit. on p. 9).

[28] F. Hutter, H. H. Hoos, and K. Leyton-Brown, "Sequential model-based optimization for general algorithm configuration," in *Learning and Intelligent Optimization – 5th International Conference, LION 5, Rome, Italy, January 17-21, 2011. Selected Papers*, C. A. C. Coello, Ed., ser. Lecture Notes in Computer Science, vol. 6683, Springer, 2011, pp. 507–523. DOI: 10.1007/978-3-642-25566-3_40 (cit. on p. 29).

[29] P. Jégou and C. Terrioux, "Tree-decompositions with connected clusters for solving constraint networks," in *Principles and Practice of Constraint Programming*, Springer, 2014, pp. 407–423 (cit. on p. 9).

[30] D. S. Johnson, "A theoretician's guide to the experimental analysis of algorithms," in *Proceedings of the 5th and 6th DIMACS Implementation Challenges*, Goldwasser, Johnson, and McGeoch, Eds., AT&T Labs – Research, American Mathematical Society, 2002, pp. 215–250 (cit. on p. 58).

[31] L. Jourdan, M. Basseur, and E.-G. Talbi, "Hybridizing exact methods and metaheuristics: a taxonomy," *European Journal of Operational Research*, vol. 199, no. 3, pp. 620–629, 2009. DOI: 10.1016/j.ejor.2007.07.035 (cit. on p. 13).

[32] U. Kjærulff, "Optimal decomposition of probabilistic networks by simulated annealing," *Statistics and Computing*, vol. 2, no. 1, pp. 7–17, 1992. DOI: 10.1007/BF01890544 (cit. on p. 11).

[33] A. M. C. A. Koster, H. L. Bodlaender, and S. P. M. Van Hoesel, "Treewidth: computational experiments," *Electronic Notes in Discrete Mathematics*, pp. 54–57, 2001 (cit. on pp. 9, 10).

[34] A. M. C. A. Koster, S. P. M. van Hoesel, and A. W. J. Kolen, "Optimal solutions for frequency assignment problems via tree decomposition," in *Graph-Theoretic Concepts in Computer Science*, ser. Lecture Notes in Computer Science, P. Widmayer, G. Neyer, and S. Eidenbenz, Eds., vol. 1665, Springer Berlin Heidelberg, 1999, pp. 338–350. DOI: 10.1007/3-540-46784-X_32 (cit. on p. 9).

[35] A. M. C. A. Koster, S. P. M. van Hoesel, and A. W. J. Kolen, "Solving partial constraint satisfaction problems with tree decomposition," *Networks*, vol. 40, no. 3, pp. 170–180, 2002. DOI: 10.1002/net.10046 (cit. on p. 9).

[36] N. Krasnogor and J. Smith, "A memetic algorithm with self-adaptive local search: TSP as a case study," in *GECCO*, 2000, pp. 987–994 (cit. on p. 13).

[37] J. Lagergren, "Efficient parallel algorithms for graphs of bounded tree-width," *Journal of Algorithms*, vol. 20, no. 1, pp. 20–44, 1996. DOI: 10.1006/jagm.1996.0002 (cit. on p. 10).

[38] P. Larrañaga, C. M. H. Kuijpers, M. Poza, and R. H. Murga, "Decomposing bayesian networks: triangulation of the moral graph with genetic algorithms," *Statistics and Computing*, vol. 7, no. 1, pp. 19–34, 1997. DOI: 10.1023/A:1018553211613 (cit. on p. 11).

[39] S. L. Lauritzen and D. J. Spiegelhalter, "Local computations with probabilities on graphical structures and their application to expert systems," *Journal of the Royal Statistical Society. Series B (Methodological)*, vol. 50, no. 2, pp. 157–224, 1988 (cit. on p. 9).

[40] Z. Lü and J.-K. Hao, "A memetic algorithm for graph coloring," *European Journal of Operational Research*, vol. 203, no. 1, pp. 241–250, 2010. DOI: 10.1016/j.ejor.2009.07.016 (cit. on p. 13).

[41] P. Merz and B. Freisleben, "A comparison of memetic algorithms, tabu search, and ant colonies for the quadratic assignment problem," in *On Evolutionary Computation, 1999. CEC 99. Proceedings of the 1999 Congress*, vol. 3, 1999, pp. 2063–2070. DOI: `10.1109/CEC.1999.785529` (cit. on p. 13).

[42] Z. Michalewicz, *Genetic Algorithms + Data Structures = Evolution Programs*, ser. Artificial Intelligence. Berlin: Springer, 1992 (cit. on p. 23).

[43] P. Moscato, "On evolution, search, optimization, genetic algorithms and martial arts: towards memetic algorithms," *Caltech concurrent computation program, C3P Report*, vol. 158-79, 1989 (cit. on pp. 12, 15, 97).

[44] P. Moscato and C. Cotta, "A modern introduction to memetic algorithms," in *Handbook of Metaheuristics*, Springer, 2010, pp. 141–183 (cit. on p. 12).

[45] N. Musliu, "An iterative heuristic algorithm for tree decomposition," in *Studies in Computational Intelligence*, C. Cotta and J. van Hemert, Eds., vol. 153, Springer, 2008, pp. 133–150 (cit. on pp. 11, 15, 18, 25, 55, 83).

[46] N. Musliu and W. Schafhauser, "Genetic algorithms for generalised hypertree decompositions," *European Journal of Industrial Engineering*, vol. 1, no. 3, pp. 317–340, 2007 (cit. on pp. 11, 15, 19, 23, 25, 55, 83).

[47] V. Nannen, S. K. Smit, and A. E. Eiben, "Costs and benefits of tuning parameters of evolutionary algorithms," *Parallel Problem Solving from Nature - PPSN X*, pp. 528–538, 2008 (cit. on p. 23).

[48] C. H. Papadimitriou, *Computational Complexity.* Academic Internet Publ., 2007 (cit. on p. 2).

[49] J. Pearson and P. G. Jeavons, "A survey of tractable constraint satisfaction problems," Royal Holloway, University of London, Tech. Rep. CSD-TR-97-15, 1997 (cit. on p. 2).

[50] R. L. Rardin and R. Uzsoy, "Experimental evaluation of heuristic optimization algorithms: a tutorial," *Journal of Heuristics*, no. 7, pp. 261–304, 2001 (cit. on p. 58).

[51] N. Robertson and P. D. Seymour, "Graph minors. II. algorithmic aspects of tree-width," *Journal of Algorithms*, vol. 7, no. 3, pp. 309–322, 1986. DOI: `10.1016/0196-6774(86)90023-4` (cit. on p. 9).

[52] S. J. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*, 2nd ed. Pearson Education, 2003 (cit. on p. 3).

[53] W. Schafhauser, "New heuristic methods for tree decomposition and generalized hypertree decompositions," Master's thesis, Vienna University of Technology, 2006 (cit. on pp. 11, 15, 19, 23, 25, 55, 83).

[54] K. Sörensen, “Metaheuristics – the metaphor exposed,” *International Transactions in Operational Research*, 2013. DOI: 10.1111/itor.12001 (cit. on p. 11).

[55] K. Sörensen and F. W. Glover, “Metaheuristics,” in *Encyclopedia of Operations Research and Management Science*, Springer, 2013, pp. 960–970 (cit. on pp. 11–13).

[56] T. G. Stützle, “Local search algorithms for combinatorial problems, Analysis, improvements, and new applications,” PhD thesis, Technische Universität Darmstadt, 1998 (cit. on p. 11).

[57] G. Syswerda, “A study of reproduction in generational and steady-state genetic algorithms,” *Foundation of Genetic Algorithms*, pp. 94–101, 1991 (cit. on p. 19).

[58] R. E. Tarjan and M. Yannakakis, “Simple linear-time algorithms to test chordality of graphs, test acyclicity of hypergraphs, and selectively reduce acyclic hypergraphs,” *SIAM Journal on computing*, vol. 13, no. 3, pp. 566–579, 1984 (cit. on p. 10).

[59] M. Widl, “Memetic algorithms for break scheduling,” Master’s thesis, Vienna University of Technology, 2010 (cit. on pp. 13, 23, 97).

[60] M. Widl and N. Musliu, “The break scheduling problem: complexity results and practical algorithms,” *Memetic Computing*, vol. 6, no. 2, pp. 97–112, 2014. DOI: 10.1007/s12293-014-0131-0 (cit. on pp. 13, 23, 97).

[61] G. J. Woeginger, “Exact algorithms for NP-hard problems: a survey,” in *Combinatorial Optimization—Eureka, You Shrink!* Springer, 2003, pp. 185–207 (cit. on p. 10).