PhD Dissertation

International Doctorate School in Information and Communication Technologies

DISI - University of Trento

Analysis of Reactive Search Optimisation Techniques for the Maximum Clique Problem and Applications

Franco Mascia

Advisor:
Dr. Mauro Brunato
Università degli Studi di Trento

December 2010

Abstract

This thesis introduces analysis tools for improving the current state of the art of heuristics for the Maximum Clique (MC) problem. The analysis focusses on algorithmic building blocks, on their contribution in solving hard instances of the MC problem, and on the development of new tools for the visualisation of search landscapes. As a result of the analysis on the algorithmic building blocks, we re-engineer an existing Reactive Local Search heuristic for the Maximum Clique (RLS–MC). We propose implementation and algorithmic improvements over the original RLS–MC aimed at faster restarts and greater diversification. The newly designed algorithm (RLS–LTM) is one order of magnitude faster than the original RLS–MC on some benchmark instances; but the proposed algorithmic changes impact also on the dynamically adjusted tabu tenure, which grows wildly on some hard instances. A more in depth analysis of the search dynamics of RLS–MC and RLS–LTM reveals the reasons behind the tabu tenure explosion and sheds some new light on the reactive mechanism. We design and implement RLS–fast which cures the issues with the tabu tenure explosion in RLS–LTM while retaining the performance improvement over RLS–MC. Moreover, building on the knowledge gained from the analysis, we propose a new hyper-heuristic which defines the new state of the art, and a novel supervised clustering technique based on a clique-finding component.

Keywords

[Combinatorial Optimisation, Maximum Clique, Stochastic Local Search, Reactive Local Search, Analysis]

Ai miei genitori.

Acknowledgements

My greatest thanks go to my friend and advisor Mauro Brunato, for the uncountable things he taught me. I can not imagine how different (read worse) my PhD studies would have been, without being contaminated with his passion for precision, dedication, and love for his work, without his patience and constant encouragement, and without such pleasant and exciting working atmosphere.

I want also to thank the coauthors of my research and publications: Mauro Brunato again, Andrea Passerini, Elisa Cilia, Roberto Battiti, Wayne Pullan, Antonio Masegosa Arredondo, David Pelta, and Marco Chiarandini.

Special thanks go to Thomas Stützle, Holger H. Hoos, Andrea Passerini and Wayne Pullan for giving me the opportunity to appreciate them both as teachers and collaborators.

I want to thank Thomas Stützle, Mauro Birattari and all the people at IRIDIA for the pleasant and fruitful time spent at ULB.

I would also want to thank Paolo Campigotto, if and only if he thanked me in his thesis.

This thesis is dedicated to my parents, I want to thank them for their love and support during these years.

A special thank goes to Elisa Cilia, the best PhD and life companion I could ever dream of.

Contents

List of Algorithms

List of Tables

List of Figures

Chapter 1

Introduction

Optimisation problems arise from virtually all areas of science and engineering, and are often characterised by a large number of variables. In combinatorial optimisation problems the optimum solution has to be sought among a discrete but possibly very large set of feasible solution. The number of these solutions can grow exponentially with the size of the problem, rendering an exhaustive search not feasible even for relatively small real world instances.

Worst-case analysis of the hardest among these problems provides us with strong theoretical results that tell us that in some case no efficient algorithms can be devised to solve arbitrary instances of the problem exactly. Therefore, a vast literature has been produced to propose and study heuristic techniques, which are able to find reasonably good solutions to real-world sized instances in polynomial time.

In spite of the massive amount of experimental data that can be collected by testing these heuristics, the intrinsic complexity of many of these techniques renders exceptionally difficult to understand thoroughly their dynamics. Therefore, the need arises for the development of techniques and tools that help understanding the complex behaviour of such heuristics and their dependence on configuration parameters, and for the description of their mutual interaction when they are assembled together forming a so-called ‘hyper-heuristic’.

The Maximum Clique Problem is a combinatorial optimisation problem that asks to find the biggest completely connected component of a graph. It has relevant applications in information retrieval, computer vision, social network analysis, computational biochemistry, bio-informatics and genomics. Its generalisations can be mapped on even more clustering related problems.

This thesis introduces analysis tools for improving the current state of the art of heuristics for the Maximum Clique (MC) problem. The analysis focusses on algorithmic building blocks, on their contribution in solving hard instances of the MC problem, and on the development of new tools for the visualisation of search landscapes. As a result of the analysis on the algorithmic building blocks, we re-engineer an existing Reactive Local Search algorithm improving its performance by an order of magnitude on large instances. Moreover, building on the knowledge gained from the analysis, we propose a new hyper-heuristic which defines the new state of the art, and a novel supervised clustering technique based on a clique-finding component.

Problem Definition The MC problem is a combinatorial optimisation problem that asks for finding the largest subset of vertices of a graph that are all pairwise adjacent.

Notation Let $G \equiv (V, E)$ be an undirected graph with a finite set of vertices $V = \{1, 2, \ldots, n\}$ and a set of edges $E \subseteq V \times V$. $G[S] = (S, E \cap S \times S)$ is the subgraph induced by the subset $S \in V$ on G, i.e. a subgraph in which $\forall\ u, v \in S$ there exists an edge between u and v if and only if $\{u, v\} \in E$. A graph $G = (V, E)$ is complete if all edges are pairwise adjacent, i.e., $E = V \times V$. A clique is a complete graph.

NP-hardness Let $G = (V, E)$ be an undirected graph and $K \leq |V|$ a positive integer. The CLIQUE problem asks if there exists a set $S \subseteq V$ such that $|S| \geq K$, and $G[S]$ is complete. Its search version asks to find the clique of maximum cardinality. CLIQUE is one of the twenty-one NP-complete problems described originally by Karp in [40], therefore its search version is NP-hard.

The reduction in [40] goes like follows. From a SATISFIABILITY instance a graph is constructed by adding a vertex for every instance of a literal appearing in each clause. Two vertices are connected by an edge if the literals appear in two different clauses and they do not contradict each other. If the SATISFIABILITY instance has K clauses then there exists a truth assignment for the formula if and only if there exists a clique of cardinality K in the graph. Figure 1.1 shows an example of this transformation. A truth assignment that satisfies the formula in Figure 1.1 is given by setting true the literals corresponding to nodes of the clique.

SATISFIABILITY has been shown by Cook [25] and independently by Levin [43] to be NP-Complete. For an introduction to complexity and intractability, and an extensive list of NP-Complete problems see [31].

Inapproximability The NP-hardness of the problem tells us that, unless P = NP, there is no hope to find efficient algorithms for solving arbitrary instances of the MC problem exactly. Even approximate solutions are hard: Håstad shows in [34] that unless NP = ZPP, CLIQUE is not approximable within $n^{1-\epsilon}$.

Generalisations Among the possible generalisations of the problem, the most relevant for our research are the quasi-clique (see for example [17]),

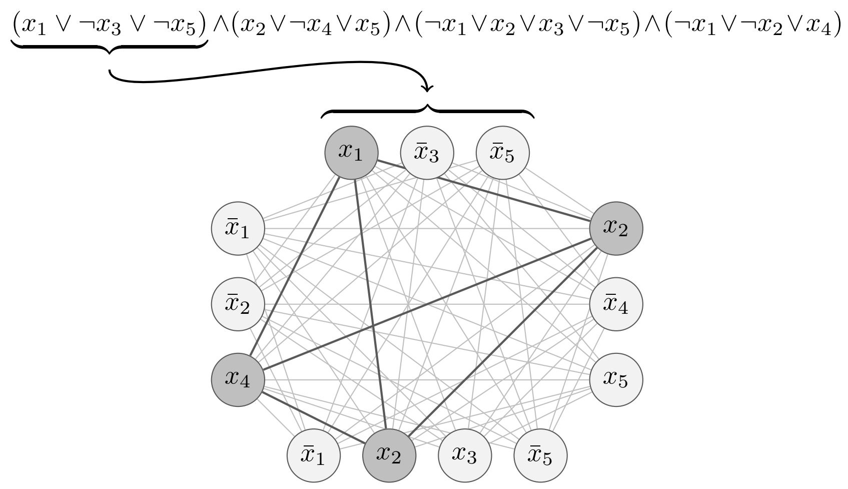

$(x_1 \vee \neg x_3 \vee \neg x_5) \wedge (x_2 \vee \neg x_4 \vee x_5) \wedge (\neg x_1 \vee x_2 \vee x_3 \vee \neg x_5) \wedge (\neg x_1 \vee \neg x_2 \vee x_4)$
x_1
$\bar{x}_3$
$\bar{x}_5$
x_2
$\bar{x}_4$
x_5
$\bar{x}_5$
x_3
x_2
$\bar{x}_1$
x_4
$\bar{x}_2$
$\bar{x}_1$

Figure 1.1: An instance of SATISFIABILITY transformed into an instance of CLIQUE. The boolean formula is satisfiable, i.e., there is a clique of cardinality greater or equal the number of clauses.

and the weighted MC where vertices and/or edges are weighted. Let $G \equiv (V, E, F)$ be a weighted undirected graph where V is the vertex set, E the edge set, and F is a function from $V \times V$ to $\mathbb{N}$ that maps an edge $e \in E$ to an integer weight. The Edge-Weighted MC Problem (WMC) requires to find the clique in V with the maximum weight:

$$V'_{\max} = \underset{\substack{V' \subseteq V \\ V' \text{ clique in } G}}{\arg\max} \sum_{u,v \in V'} F(u,v).$$

Being a generalisation of the MC problem, the edge-weighted version is also NP-hard.

Another possible generalisation can be obtained by relaxing the constraint on the connectivity of the nodes belonging to the clique. Given an undirected graph (V, E), and two parameters λ and γ with $0 \leq \lambda \leq \gamma \leq 1$, the subgraph induced by a subset of the node set $V' \subseteq V$ is a (λ, γ)-*quasi-*

clique if, and only if, the following two conditions hold:

$$\begin{aligned} \forall v \in V' : \quad \deg_{V'}(v) &\geq \lambda \cdot (|V'| - 1) && (1.1) \\ |E'| &\geq \gamma \cdot \binom{|V'|}{2}, && (1.2) \end{aligned}$$

where $E' = E \cap (V' \times V')$ and $\deg_{V'}(v)$ is the number of elements of V' connected to v.

1.1 State of the Art and Related Works

The MC problem is a prominent and well studied combinatorial optimisation problem. Among the Stochastic Local Search algorithms that have been proposed in the literature, we cite Deep Adaptive Greedy Search (DAGS) [32] that uses an iterated greedy construction procedure with vertex weights; the k-opt heuristic [41] that is based on a conceptually simple Variable Depth Search (VDS) procedure; VNS [37] that is a basic variable neighbourhood search heuristic that combines greedy search with simplical vertex tests in its descent steps; Reactive Local Search [13] (RLS) and Dynamic Local Search [54] (DLS) which will be described later; and Phased Local Search [53], which sequentially cycles through greedy vertex degree selection, random selection and vertex penalty based selection heuristics. The theoretical grounds for the analysis of Stochastic Local Search (SLS) can be found in [38] and [10]. Among the evolutionary algorithms we cite [44], which proposes a combination of local search and Genetic Algorithms for escaping from local optima; and [58], which proposes and analyses four variants of an Ant Colony Optimisation algorithm for the MC.

In this thesis the focus will be on two of these heuristics, namely RLS and DLS. Reactive Search [14, 9] advocates the use of machine learning to automate the parameter tuning process and make it an integral and fully documented part of the algorithm. Learning is performed on-line, therefore task-dependent and local properties of the configuration space can be used. A Reactive Local Search algorithm for the MC problem (RLS–MC) is proposed in [12, 13]. RLS–MC is based on tabu search complemented by a history-sensitive feedback scheme to determine the amount of diversification. The reaction acts on the prohibition tenure parameter that decides the temporary prohibition of selected moves in the neighbourhood. The performance obtained by [13] in computational tests appears to be significantly better with respect to all algorithms tested at the second DIMACS implementation challenge (1992-93).

The Dynamic Local Search algorithm for MC (DLS–MC) has been proposed in 2006 [54]. It is based on a clique expansion phase followed by a plateau search after a maximal clique is encountered. Diversification is ensured by the introduction of vertex 'penalties' that change the contribution of individual vertices to the objective function. Such penalties are increased on vertices that appear in a clique and gradually decreased in time. The frequency at which the penalties are decreased is controlled by a 'penalty delay' parameter, which has been manually tuned for families and sometimes even subfamilies of instances. The performances of DLS–MC are highly dependent on the appropriate penalty delay value for the instance at hand.

While most of this thesis is dedicated to the analysis of the contribution of different building blocks to the performance of local search algorithms for the MC problem and to the study of the dynamics of RLS, the introduction of new tools for visualising search landscapes helps the intuition behind the hardness of specific instance classes.

Work that combines visualisation and optimisation dates back to [51], where multidimensional scaling and other techniques are applied to the visualisation of evolutionary algorithms, while other contributions are aimed at human-guided search [1] where the computer finds local optima by hill-climbing while the user identifies promising regions of the search space. Visualisation of Pareto sets in Evolutionary Multi-Objective optimisation is investigated in [42] by finding a mapping which maintains most of the dominance relationships. In [36] the authors propose a visualisation suite for designing and tuning SLS algorithms. Starting from a selection of candidate solutions, the visualisation tool uses a spring-based layout scheme to represent the solutions in two dimensions. The algorithm execution traces are then represented as trajectories around the laid out solutions, and the resulting analysis is used by the researchers to tune the algorithm studied.

1.2 Analysis and Re-engineering of Algorithms

This section reports the main contribution of this thesis, both in terms of analysis methodology and efficient algorithm implementation. The applications will be discussed in Section 1.3.

1.2.1 Analysis of Algorithmic Building Blocks

Chapter 2 describes a methodology to isolate and study the different algorithmic components used by two of the aforementioned techniques, RLS–MC and DLS–MC, with the aim to gain insights on the contribution of the single components to the algorithm performance both in terms of solution quality and run-times. The analysis focusses, in particular, on the dynamics introduced in the behaviour of the local search algorithms by three paradigmatic methods aimed at achieving a proper balance i) using prohibitions to achieve diversification and avoid small cycles in the search trajectory (limit cycles or the equivalent of ‘chaotic attractors’ in discrete dynamical systems), ii) using restarts triggered by events happening during the search, and iii) using modifications of the objective function to influence the trajectory and achieve diversification by modifying the fitness surface instead of reducing the number of admissible (non-prohibited) moves. In particular, the investigation considers the effects of using the vertex degree information during the search.

1.2.2 Search Landscape Visualisation

As a useful complement to the analytical work, in Chapter 3 we propose a set of techniques for the visualisation of search landscapes aimed at supporting the researcher's intuition on the behaviour of a SLS algorithm applied to a combinatorial optimisation problem. We discuss the scalability issues posed to visualisation by the size of the problems and by the number of potential solutions, and we propose approximate techniques to overcome them. The proposed visualisation technique is also capable to rendering explicitly the geographic metaphors used by researchers to describe areas of interest of the landscape, and has therefore a tutorial valence.

1.2.3 Engineering an Efficient Algorithm

The analysis of the algorithmic building blocks described above tells just part of the story. A comprehensive work on algorithmic efficiency needs also considering low level implementation details, choices of data structures, programming languages, and knowledge on how the compiler optimises the code.

Building on the results of the empirical analysis and from the profiling of various algorithm implementations, we propose a new algorithm that presents two kinds of changes with respect to the original RLS version. The first changes are algorithmic and influence the search trajectory, while the second one refers only to the more efficient implementation of the supporting data structures, with no effect on the dynamics. In the previous version of RLS, the search history was cleared at each restart, now, in order to allow for a more efficient diversification, the entire search history is kept in memory. To underline this fact, the new version is called RLS 'Long Term Memory', or RLS–LTM. However, having a longer memory causes the tabu tenure T to explode on some specific instances characterised by many repeated configurations during the search; in fact, if the prohibition becomes much larger than the current clique size, after a maximal clique is encountered and one node has to be extracted from the clique, all other nodes will be forced to leave the clique before the first node is allowed to enter again. This may cause spurious oscillations in the clique membership which may prevent discovering the globally optimal clique. An effective way to avoid the above problem is to introduce an upper bound to the tenure equal to a proportion of the current estimate of the maximum clique.

The improvement in the steps per seconds achieved by RLS–LTM over the original RLS increases with the graph dimension reaching a factor of 22 for graphs with thousands nodes, see Chapter 4 for more details. The total computational cost for solving a problem is the product of the number of iterations times the cost of each iteration. More complex algorithms like RLS risk that the higher cost per iteration is not compensated by a sufficient reduction in the total number of iterations. In RLS, a single operation can be the addition or the removal of a node from the current configuration. To this aim, we propose an empirical model for the CPU time per iteration which is linear in the graph size $n = |V|$, and the degree in the complementary graph $\overline{G}$ of the node that has been added or removed in the operation:

$$T(n, deg_{\overline{G}}) = \alpha\ n + \beta\ deg_{\overline{G}} + \gamma\ .$$

Fitting the model on our testing machine we get:

$$T(n, deg_{\overline{G}}) = 0.0010\ n + 0.0107\ deg_{\overline{G}} + 0.0494\ .$$

Let us note that the cost for using the history data structure, which is approximately included in the constant term in the above expression, becomes rapidly negligible as soon as the graph dimension and density are not very small. In fact the memory access costs approximately less than 50 nanoseconds per iteration while the total cost reaches rapidly tens of microseconds in the benchmark instances. These results drastically change the overall competitiveness of the RLS technique.

1.2.4 A Comparison of Tabu Search Variations

A more focussed and extensive analysis of the Reactive Tabu Search (RTS) algorithm and of the datasets reveals that for almost every instance of the DIMACS benchmark set, there is a narrow range of good values that can be assumed by the prohibition parameter T of a Tabu Search heuristics. Outside this narrow range, the time needed by the heuristic to converge to good quality solutions increases significantly.

What emerges from the study is that RTS is quite effective in the online tuning of the tabu tenure, i.e., it quickly converges to good values for the parameter with very little computational overhead. There is no clue, at least not on the instances taken in consideration, that the reaction mechanism is influenced by local characteristic of the instance. In fact, uniform random updates of the parameter lead to results that are statistically indistinguishable from RTS. We can say that a Robust Tabu Search heuristic [60] (RoTS) behaves like RTS, provided that RoTS is made to operate on a range of values appropriate for the instance at hand.

Part of the analysis is also devoted to the study of the specific RLS–LTM dynamics. The insights obtained led to the implementation of a new version of RLS that shares the same performance of RLS–LTM but not its issues with the convergence of the prohibition parameter discussed in Chapter 4.

1.3 Applications

The MC problem is a paradigmatic combinatorial optimisation problem with relevant applications [48, 4], including information retrieval, computer vision, and social network analysis. Recent interest includes computational biochemistry, bio-informatics and genomics, see for example [39, 20, 50].

The outcomes of the analysis techniques presented in Section 1.2 are of two kinds. In Section 1.3.1 we introduce a new hyper heuristic for the MC, while in Section 1.3.2 we describe a different type of application, i.e., how a heuristics for the MC can be used in the context of learning with structured output.

1.3.1 Cooperating Local Search

The analysis of the search dynamics and the improved RLS–LTM implementation described in the previous chapters led naturally to the design of a new hyper-heuristic for the MC. The relatively recent explosion of availability of multi-core desktop and laptop computers has made the case for the design of a parallel hyper-heuristic. Chapter 6 presents the research that lead to the design and implementation of Cooperating Local Search (CLS) for the MC problem.

CLS controls several copies of four low level heuristics that are the most effective for the MC problem. Communication between the low level heuristics allows to have truly complementary heuristics that focus on different parts of the search space. Moreover, CLS dynamically reallocates copies of the four heuristics to CPU cores, in order to ensure that the most effective mix of low level heuristics for the instance at hand is used. CLS performance is comparable and sometimes improves over single heuristic optimised for specific instances of the DIMACS benchmark dataset.

1.3.2 Supervised Clustering

As an application of the study in Section 1.2, we tackle the problem of predicting the set of residues of a protein that are involved into the binding of metal ions and more generally participating in active sites. In this introduction, we will try to abstract as much as possible from the specific biological problem and omit why predicting active sites and metal binding sites is fundamental for understanding the protein function. All the details about the specific problem and the results achieved can be found in Chapter 7. We formulate the problem as learning with structured output.

Let us consider a protein as a string in Σ^* over a small alphabet Σ. We want to learn a function that given a string $s \in \Sigma^*$ maps it to a partial clustering $\mathcal{Y}$, i.e., a set of disjoint subsets of the positions in s, where each subset contains the elements in the string that belong to the same active site, or that cooperate in the binding of a metal ion. More formally, if the length of the string is l, we define the set of all possible partial clusterings of the l indices in the following way:

$$\mathcal{C}_l = \Big\{ \mathcal{Y} \subseteq \mathcal{P}(\{1, \ldots, l\}) : \bigcup \mathcal{Y} \subseteq \{1, \ldots, l\} \wedge \\ (A, B \in \mathcal{Y}\ A \neq B \Rightarrow A \cap B = \emptyset) \wedge \emptyset \notin \mathcal{Y} \Big\}.$$

The set $\mathcal{C}_l$ is finite since $\mathcal{C}_l \subseteq \mathcal{P}(\mathcal{P}(\{1, \ldots, l\}))$. We define the set $\mathcal{C}$ as:

$$\mathcal{C} = \bigcup_{l \in \mathbb{N}} \mathcal{C}_l.$$

Given an example set of known mappings:

$$S \subseteq \Sigma^* \times \mathcal{C},$$

such that

$$[(s, \mathcal{Y}), (s', \mathcal{Y}') \in S \wedge \mathcal{Y} \neq \mathcal{Y}'] \Rightarrow s \neq s', \qquad (s, \mathcal{Y}) \in S \Rightarrow \mathcal{Y} \in \mathcal{C}_{|s|},$$

we want to learn a function:

$$f : \Sigma^* \to \mathcal{C}, \tag{1.3}$$

such that

$$(s, \mathcal{Y}) \in S \Rightarrow f(s) = \mathcal{Y}, \qquad f(s) \in \mathcal{C}_{|s|}.$$

We split the problem of predicting with structured output in two parts: first we learn a pairwise similarity measure on the elements in each cluster with a binary classifier, then we use it to construct a edge-weighted graph and mine the clusters as weighted MC. We use as positive examples all pairs of elements which belong to the same cluster. Hence, the training set is:

$$S' \subseteq \Sigma^* \times \mathbb{N} \times \mathbb{N} \times \{\pm 1\},$$

where

$$\begin{gathered}(s, u, v, y) \in S' \Leftrightarrow \\ u, v \in \{1, \ldots, |s|\} \\ \wedge \exists \mathcal{Y} \in \mathcal{C}_{|s|} : [(s, \mathcal{Y}) \in S \wedge (y = +1 \Leftrightarrow u \neq v \wedge \exists C \in \mathcal{Y}\ \{u, v\} \in C)].\end{gathered}$$

We use S' to train a support vector machine SVM : $\Sigma^* \times \mathbb{N} \times \mathbb{N} \to \mathbb{R}$ that given a string s and pair u, v returns a score that is a confidence that u, v belong to the same cluster. In our case the score is the distance from the margin of the trained classifier.

Given a string $s \in \Sigma^*$, we use the trained SVM margin function to build a weighted graph $G^s = (V^s, E^s, F^s)$, where $V^s = \{1, \ldots, |s|\}$, $F^s(u, v) = t(\text{SVM}(s, u, v))$ with t being a scaling function in a suitable range, and $E^s(e^s_{u,v})$ being the adjacency matrix where:

$$e^s_{u,v} = \begin{cases} 1 & \text{if } \text{SVM}(s, u, v) \geq \theta \\ 0 & \text{otherwise} \end{cases}$$

and θ is a suitable threshold value. By construction, it is clear that cliques with the highest weight in G^s correspond to the desired clusters in s. The threshold θ accounts for errors in the prediction of the weights on the edges.

The novel distance-based supervised clustering approach improves substantially over the only other existing approach in predicting metal-binding sites from sequence to our knowledge [29]. The algorithm naturally handles the lack of knowledge in the number of clusters, partial clusterings with many outliers, and it can also be applied in the setting of overlapping clusters. Significant improvements over the state of the art are also obtained in predicting active sites from the protein 3D structure.

1.4 Publications in Connection with this Thesis

Part of the work presented in this thesis is based on several publications by the authors and co-authors.

- Chapter 2 and 4 present the analysis of algorithmic building blocks and a new implementation of RLS–MC. These chapters are based on:

  Roberto Battiti and Franco Mascia. Reactive and dynamic local search for max-clique: Engineering effective building blocks. *Computers & Operations Research*, 37(3):534–542, March 2010.

- The search visualisation techniques described in Chapter 3 are based on:

  Franco Mascia and Mauro Brunato. Techniques and tools for local search landscape visualization and analysis. In *Proceedings of SLS 2009, Engineering Stochastic Local Search Algorithms. Designing, Implementing and Analyzing Effec-*

tive Heuristics, International Workshop, Brussels, Belgium, pages 92–104, 2009.

- Chapter 6 describes a novel hyper-heuristic for the MC problem. This chapter is based on:

  Wayne J. Pullan, Franco Mascia, and Mauro Brunato. Cooperating local search for the maximum clique problem. *Journal of Heuristics*, 2010.

- The application of a weighted maximum clique heuristic to a supervised clustering problem presented in Chpater 7 is based on:

  Franco Mascia, Elisa Cilia, Mauro Brunato, and Andrea Passerini. Predicting Structural and Functional Sites in Proteins by Searching for Maximum-Weight Cliques. In *Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence (AAAI-10)*, pages 1274–1279. AAAI Press, July 2010.

- Finally Chapter 5 describes ongoing work that I started while at ULB, and which will be published as soon as experimental data are finalised.

Other publications not directly reported in this thesis, but arising from the work during the PhD are [2, 10, 23].

Acknowledgements

Part of this work has been conducted while visiting the Insitut de Recherches Interdisciplinaires et de Développements en Intelligence Artificielle (IRIDIA), Univeristé Libre de Bruxelles.

Part of this work has been supported by project BIONETS (FP6-027748) funded by the FET program of the European Commission.

Many thanks to Michèle Sebag for very fruitful discussions.

We thank Holger Hoos and Wayne Pullan for making available the software corresponding to the DLS–MC algorithm.

Part I

Analysis and Re-engineering of Algorithms

Chapter 2

Analysis of Algorithmic Building Blocks

This chapter describes a methodology [11] to isolate and study the different algorithmic components used by RLS and DLS–MC, with the aim to gain insights on the contribution of the single components to the algorithm performance both in terms of solution quality and run-times.

2.1 Prohibition- and Penalty-based Methods

The availability of heuristic solution techniques for relevant combinatorial problems is now large and the scientific and practical issues arise of tuning, adapting, combining and hybridising the different techniques. In hybrid meta-heuristics, the potential for reaching either better average results, or more robust results with less variability is large, but the task in complex. First of all, if one naively tries to consider all possible combinations of component and parameters a combinatorial explosion occurs. Secondly, like in all scientific challenges, one aims not only at beating the competition on specific benchmarks, but also at understanding the contribution of the different parts to the whole and at discriminating the basic principles for achieving successful hybrids.

The present investigation is focussed on algorithmic components used to solve the maximum clique problem in graphs with state-of-the-art results. In particular it is focussed on combining methods based on local search with memory-based complements to achieve a proper balance of intensification and diversification during the search. The considered paradigms are of: i) using prohibitions to achieve diversification and avoid small cycles in the search trajectory (limit cycles or the equivalent of 'chaotic attractors' in discrete dynamical systems), ii) using restarts triggered by events happening during the search, and iii) using modifications of the objective function to influence the trajectory and achieve diversification by modifying the fitness surface instead of reducing the number of admissible (non-prohibited) moves.

In addition, Stochastic Local Search Engineering methods are considered to develop efficient implementations of the single steps of the SLS-based techniques, by considering data structures to support the choice of the next move and the use of memory during the search. Let us briefly summarise the considered techniques and the concentration of this chapter.

Reactive Search [14, 9] advocates the use of machine learning to automate the parameter tuning process and make it an integral and fully documented part of the algorithm. Learning is performed on-line, and therefore task-dependent and local properties of the configuration space can be used. A Reactive Local Search (RLS) algorithm for the solution of the Maximum-Clique problem is proposed in [12, 13]. RLS is based on local search complemented by a feedback (history-sensitive) scheme to determine the amount of diversification. The reaction acts on the single parameter that decides the temporary prohibition of selected moves in the neighbourhood. The performance obtained in computational tests appears to be significantly better with respect to all algorithms tested at the the second DIMACS implementation challenge (1992/93)[1].

In 2006, a stochastic local search algorithm (DLS–MC) is developed in [54]. It is based on a clique expansion phase followed by a plateau search after a maximal clique is encountered. Diversification uses vertex penalties which are dynamically adjusted during the search, a ‘forgetting’ mechanism decreasing the penalties is added, and vertex degrees are not considered in the selection. The authors report a very good performance on the DIMACS instances after a preliminary extensive optimisation phase to determine the optimal penalty delay parameter for each instance. While the number of iterations (additions or deletions of nodes to the current clique) is in some cases larger than that of competing techniques, the small complexity of each iteration when the algorithm is realised through efficient supporting data structures leads to smaller overall CPU times.

The motivation of this chapter is threefold. First, we want to investigate how the different algorithmic building blocks contribute to effectively solving max-clique instances corresponding to random graphs with different statistical properties. In particular, the investigation considers the effects of using the vertex degree information during the search, starting from simple to more complex techniques. Second, we want to assess how different implementations of the supporting data structures affect CPU times. For example, it may be the case that larger CPU times are caused by using a high-level language implementation w.r.t. low-level ‘pointer arithmetic’. Having available the original software simplified the starting point for this analysis. Third, the DIMACS benchmark set (developed in 1992) has been around for more than a decade and there is a growing risk that the desire to get better and better results on the same benchmark will bias the search of algorithms in an unnatural way. We therefore decided to concentrate the experimental part on two classes of random graphs, chosen to assess

[1]http://dimacs.rutgers.edu/Challenges/

the effect of degree variability on the effectiveness of different techniques.

2.2 Building Blocks of Increasing Complexity

In local search algorithms for MC, the basic moves consist of the addition to or removal of single nodes from the current clique. A swap of nodes can be trivially decomposed into two separate moves. The local changes generate a search trajectory $X^{\{t\}}$, the current clique at different iterations t. Two sets are involved in the execution of basic moves: the set of the improving neighbours PossibleAdd which contains nodes connected to all the elements of the clique, and the set of the level neighbours OneMissing containing the nodes connected to all but one element of the clique, see Figure 2.1. The various simple building blocks considered are named following the BasicScheme–CandidateSelection structure. The BasicScheme describes how the greedy expansion and plateau search strategies are combined, possibly with prohibitions or penalties. The CandidateSelection specifies whether the vertex degree information is used during the selection of the next candidate move. If it is used, there are two possibilities: of using the static node degree in G or the dynamic degree in the subgraph induced by the PossibleAdd set.

2.2.1 Repeated Expansions

The starting point for many schemes is given by expansion of a clique after starting from an initial seed vertex. At each iteration the next vertex to be added can be chosen from the PossibleAdd set through different levels of 'greediness' when one considers the vertex degrees:

- **Exp–Rand.** The node is selected at random among the possible additions. When a maximal clique is is encountered one restarts from a random node. The pseudo-code is shown Listing 2.1.

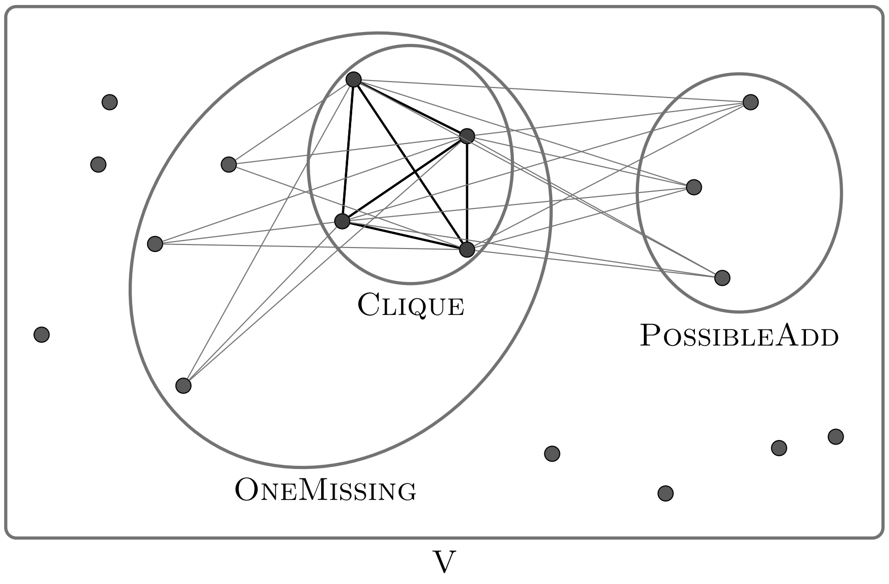


Figure 2.1: Neighbourhood of the current clique. Not all edges are depicted in the figure.

Listing 2.1: Greedy expansion algorithm.

```
function Exp–Rand ($maxIterations$)
    $iterations \leftarrow 0$
    while $iterations < maxIterations$ do
        $C \leftarrow$ random $v \in V$
        expand ($C$)

function expand ($C$)
    while PossibleAdd $\neq \emptyset$ do
        $C \leftarrow C \cup$ random $v \in$ PossibleAdd
        $iterations \leftarrow iterations + 1$
```

- **Exp–StatDegree.** At each iteration, a random node is chosen among the candidates having the highest degree in G. The EXPAND subroutine in Listing 2.1 is modified by substituting line 9 with:
  $C \leftarrow C \cup \{rand\ v \in \mathsf{PossibleAdd} : deg_G(v)\ is\ max\}$.
- **Exp–DynDegree.** In this version, the selection of the candidate is not based on the degree of the nodes in G, but on the degree in PossibleAdd. This greedy choice will maximise the number of nodes remaining in PossibleAdd after the last addition. Line 9 becomes:
  $C \leftarrow C \cup \{rand\ v \in \mathsf{PossibleAdd} : deg_{\text{POSSIBLEADD}}(v)\ is\ max\}$.

2.2.2 Expansion and Plateau Search

This algorithm alternates between a greedy expansion and a plateau phase, choosing between the possible candidate nodes with different ways to consider the vertex degrees:

- **ExpPlat–Rand.** During the expansion phase, new vertices are chosen randomly from PossibleAdd and moved to the current clique. When PossibleAdd is empty and therefore no further expansion is possible, the plateau phase starts. In this phase, a node belonging to the level neighbourhood OneMissing is swapped with the only node not connected to it in the current clique. The plateau phase does not increment the size of the current clique and it terminates as soon as there is at least an element in the PossibleAdd set, or if no candidates are available in OneMissing. As it is done in [54], nodes cannot be selected twice in the same plateau phase. In order to avoid infinite loops, the number of plateau searches is limited to maxPlateauSteps. Starting from EXP–RAND, the base algorithm is adapted to deal with the alternation of the two phases, see Listing 2.2. Let us note that, if PLATEAU returns with PossibleAdd $\neq \emptyset$ then a new expansion is tried

Listing 2.2: ExpPlat–Rand alternates between expand and plateau phases.

```
 1 function ExpPlat–Rand ($maxIterations$, $maxPlateauSteps$)
 2     $iterations \leftarrow 0$
 3     while $iterations < maxIterations$ do
 4         $C \leftarrow$ random $v \in V$
 5         while PossibleAdd $\neq \emptyset$ do
 6             expand ($C$)
 7             plateau ($C, maxPlateauSteps$)

 9 function plateau ($C$, $maxPlateauSteps$)
10     $count \leftarrow 0$
11     while PossibleAdd $= \emptyset$ and OneMissing $\neq \emptyset$
12     and $count < maxPlateauSteps$ do
13         $C \leftarrow C \cup$ random $v \in$ OneMissing
14         remove from $C$ the node not connected to $v$
15         $iterations \leftarrow iterations + 2$
16         $count \leftarrow count + 1$
```

as described in line 5–7. The iterations are incremented by 2 during a swap because it is counted as a deletion followed by an addition.

- **ExpPlat–StatDegree.** This algorithm is a modified version of ExpPlat–Rand (Listing 2.2) with the static degree selection during the expansion and the plateau.

- **ExpPlat–DynDegree.** This algorithm is the same of ExpPlat–StatDegree, apart from the selection based on the dynamic degree during the expansion phase.

2.2.3 Algorithms Based on Penalties or Prohibitions

More complex schemes can be obtained by using diversification strategies to encourage the search trajectories to visit unexplored regions of the search space. These methods are particularly effective for 'deceptive' instances [16], where the sub-optimal solutions attract the search trajectories.

- **ExpPlatProhibition–Rand.** A simple diversification strategy can be obtained by prohibiting selected moves in the neighbourhood. In detail, after a node is added or deleted from the current clique, the algorithm prohibits moving it for the next T iterations. Prohibited nodes cannot be considered among the candidates of expansion and plateau phases. When all the moves are prohibited a restart is performed.

- **DLS–MC.** To achieve diversification during the search, penalties are assigned to vertices of the graph [54]. The algorithm alternates between expansion and plateau phases. Selection is done by choosing the best candidate among the set of the nodes in the neighbourhood having minimum penalty.

  When the algorithm starts, the penalty value of every node is initialised to 0 and when no further expansion or plateau moves are possible, the penalties of nodes belonging to the clique are incremented by one. All penalties are decremented by one after *pd* (penalty delay) restarts, see [54] for additional details and results.

- **RLS.** This algorithm alternates between expansion and plateau phases, like DLS–MC, but it selects the nodes among the non-prohibited ones which have the highest degree in PossibleAdd. The prohibition time is adjusted reactively depending on the search history. In the 'history' a fingerprint of each configuration is saved in a hash-table. Restarts are executed only when the algorithm cannot improve the current configuration within a fixed number of iterations, see [13] for details.

  To allow for a comparison between different amount of 'greediness' in the node selection, a modification of RLS is introduced (called

RLS–StatDegree) which uses the static degree instead of the dynamic degree selection.

2.3 Computational Experiments

The computational experiments, presented in this chapter are on two classes of random graphs, and are aimed at comparing the different algorithmic building blocks and their impact on average number of steps required to find the maximum clique, as well as the cost per single iteration.

To ensure that hard instances are considered in the test, a preliminary study investigates the empirical hardness as a function of the graph dimension.

2.3.1 Benchmark Graphs

Performance and scalability tests are made on two different classes of random graphs:

Binomial random graphs A binomial graph $GIL(n,p)$, belonging to Gilbert's model $\mathcal{G}(n,p)$ is constructed by starting from n nodes and adding up to $\frac{n(n-1)}{2}$ undirected edges, independently with probability $0 < p < 1$. See [8] for generation details.

Preferential Attachment Model A graph instance $PAT(n,d)$, of the preferential attachment model, introduced in [5] is built by starting from d disconnected nodes and adding successively the remaining $n-d$ nodes. The edges of the the newly added nodes are connected to d existing nodes, with preferential attachment to nodes having a higher degree, i.e., with probability proportional to the number of edges present between the existing nodes.

Nodes	GIL(n, 0.3)	PAT(n, n/3)
100	6	13
200	7	19
300	8	25
400	8	31
500	8	37
600	8	42
700	9	48
800	9	54
900	9	57
1000	9	60
1100	10	64
1200	10	70
1300	10	74
1400	10	79
1500	10	86

Table 2.1: Best empirical maximum cliques in the benchmark graphs.

In binomial graphs, the degree distribution for the different nodes will be peaked on the average value, while in the preferential attachment model, the probability that a node is connected to k other nodes decreases following a power-law i.e. $P(k) \sim k^{-\gamma}$ with $\gamma > 1$.

In the experiments, the graphs are generated using the NetworkX library [35], for a number of nodes ranging from 100 to 1500. Because of the hardness of MC, the optimal solutions of large instances cannot be computed and one must resort to the empirical maximum. The empirical maximum considered in the experiments is the best clique that RLS is able to find in 10 runs of 5 million steps each. In no case DLS–MC with *pd* equal to 1 is able to find bigger cliques for the same number of iterations. The sizes of the empirical maximum cliques in the various graphs are listed in Table 2.1.

The algorithms are tested against our data set, to compute the distribution function of the iterations needed to find the empirical maximum. The maximum number of steps per iteration is set to 10 million and each test is repeated on the same graph instance 100 times. For the algorithms having a plateau phase, maxPlateauSteps is set to 100.

We count as one iteration each add- or drop-move executed on the clique. DLS–MC code was modified to count the steps in this manner, to be able to make comparisons with the other algorithms. The CPU time spent by each iteration is measured on our reference machine, having one Xeon processor at 3.4 GHz and 6 GB RAM. The operating system is a Debian GNU/Linux 3.0 with kernel 2.6.15-26-686-smp. All the algorithms are compiled with g++ compiler with '-O3 -mcpu=pentium4'.

Figure 2.2 and Figure 2.3 summarise with standard box-and-whisker plots the medians, the quartiles, and the outliers of the iterations by ExpPlat–Rand. Figure 2.2 shows that there are some instances which are significantly harder than others. The sawtooth trend of the plot is due to the fact that ExpPlat–Rand needs on average more iterations to solve instances of the Gilbert model corresponding to the increase of the expected clique size in Table 2.1. Instances become then easier when the number of nodes increases and the maximum clique remains of the same size while the number of optimal cliques increases. This is confirmed also by all other algorithms considered and explained by the theoretical results by Matula [47].

The sawtooth behaviour is hardly visible in Fig 2.3 because of the different granularity of the cliques dimension with respect to the graph sizes considered.

Figure 2.4 reports the number of iterations to find the empirical maximum clique in PAT(1100,366) graphs by the most significant techniques considered in this chapter. It can be observed that RLS achieves the best results for this class of graphs. Furthermore, the variation in the number of iterations is smaller, implying a larger robustness of the technique. Additional results and discussions are presented in Section 2.3.2.

2.3.2 Results Summary

Table 2.2 presents the results on two specific instances of random graphs: $GIL(1100, 0.3)$ and $PAT(1100, 366)$. The choice of $GIL(1100, 0.3)$ is de-

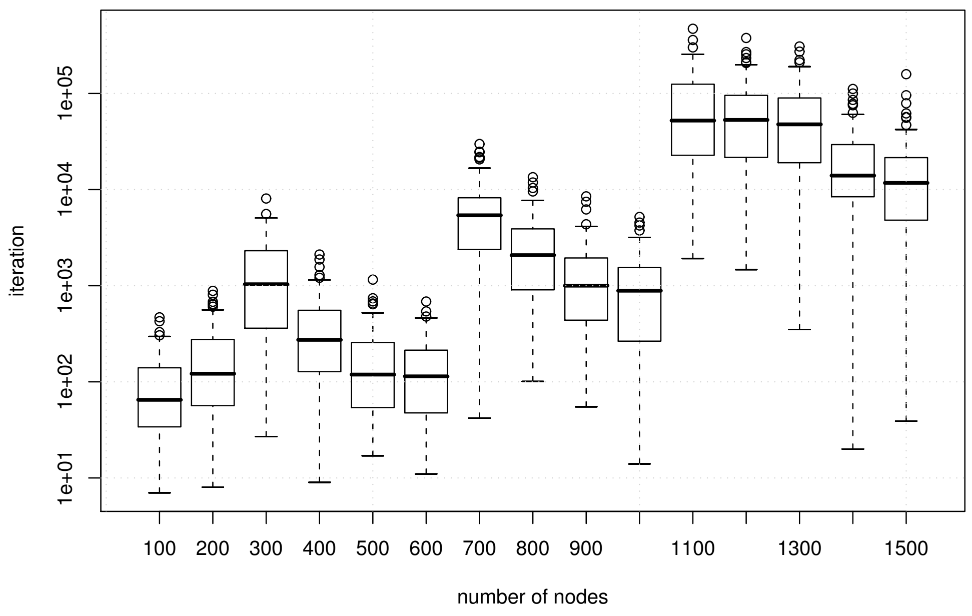


Figure 2.2: Iterations of ExpPlat–Rand to find the empirical maximum clique in $GIL(n, 0.3)$. Y axis is logarithmic.

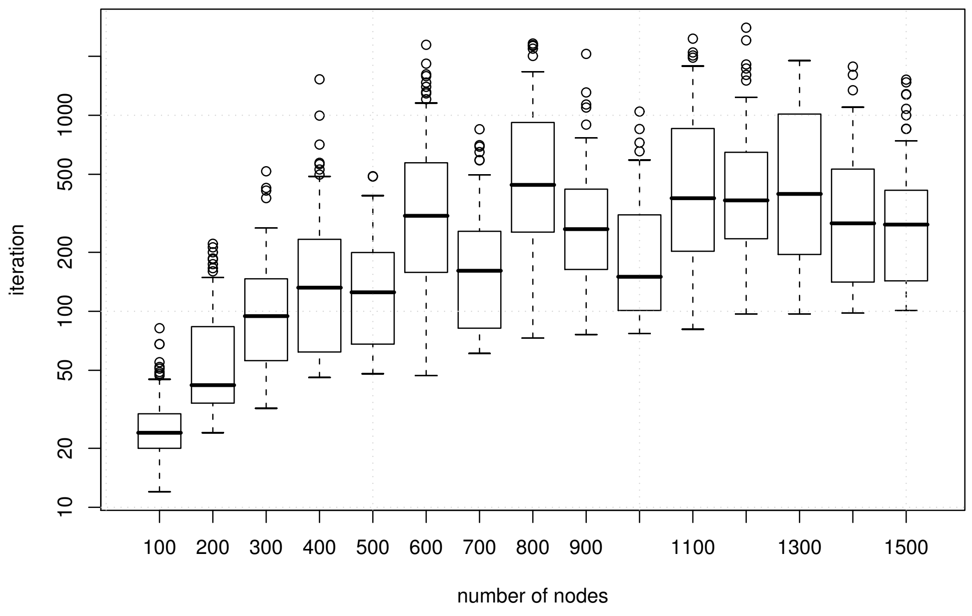


Figure 2.3: Iterations of ExpPlat–Rand to find the empirical maximum clique in $PAT(n, n/3)$. Y axis is logarithmic.

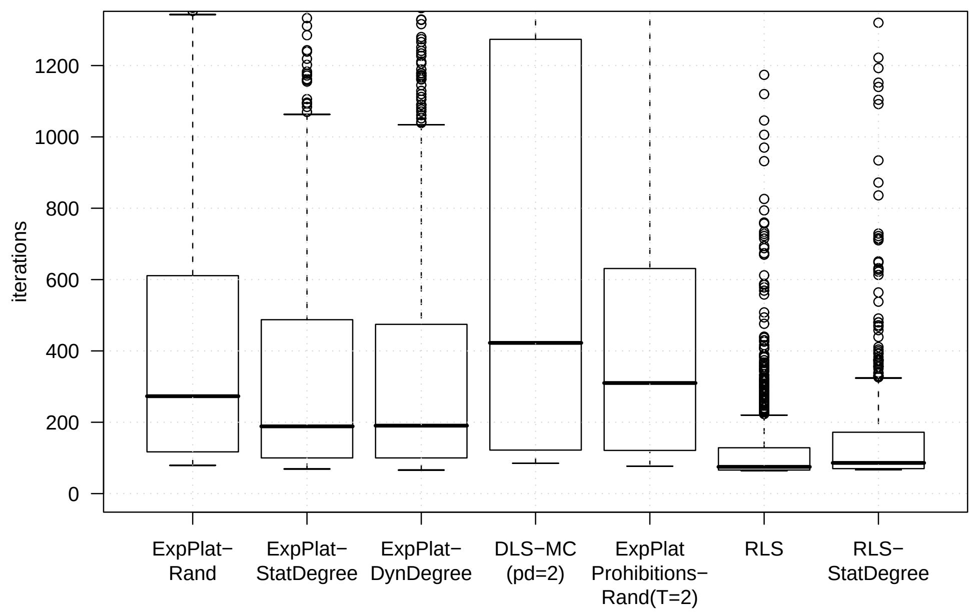


Figure 2.4: Iterations to find the empirical maximum clique in $PAT(1100, 366)$. Results for the most significant algorithms are reported.

termined by the fact that it is empirically the most difficult instance of our data set, while $PAT(1100, 366)$ is chosen with the same number of nodes. The results are for 100 runs on 10 different instances.

From Table 2.2, it is clear that algorithms based only on expansions are not always able to find the maximum clique in the given iteration bound, especially on hard instances. The plateau phase increases dramatically the success rate.

Degree consideration is effective in the Preferential Attachment model, while in Gilbert's graphs, where the nodes tend to have similar degrees, penalty- or prohibition-based algorithms win. For example the reduction in iterations achieved by ExpPlat–StatDegree over ExpPlat–Rand on $PAT(1100, 366)$ is about 31%. The results is confirmed by a Mann-Whitney U-test (Wilcoxon rank-sum test) at significance level 0.05: p-value is $3.768 \cdot 10^{-15}$. On the contrary, algorithms using degree information have poorer performances on Gilbert's graphs, if compared with their com-

Algorithm	$GIL(1100, 0.3)$			$PAT(1100, 366)$		
	Iter.	μs/Iter.	CPU (sec)	Iter.	μs/Iter.	CPU (sec)
Exp–Rand	**[92%]***	8.90	-	**[0%]***	3.20	-
Exp–StatDegree	**[0%]***	8.30	-	**[40%]***	3.10	-
Exp–DynDegree	**[10%]***	104.00	-	**[0%]***	10.3	-
ExpPlat–Rand	74697	5.80	.433	273	3.10	.00084
ExpPlat–StatDegree	**[60%]***	5.70	-	189	3.10	.00058
ExpPlat–DynDegree	75577	27.20	2.055	191	7.55	.00144
DLS–MC(pd=2)	75943	5.90	.448	423	3.20	.00135
DLS–MC(pd=4)	63467	5.90	.374	**[99%]***	3.20	-
DLS–MC(pd=8)	73831	5.90	.453	**[85%]***	3.20	-
ExpPlatPro.–Rand(T=2)	65994	5.80	.382	310	3.10	.00096
ExpPlatPro.–Rand(T=4)	67082	5.90	.395	333	3.10	.00103
ExpPlatPro.–Rand(T=8)	67329	5.80	.390	329	3.15	.00103
RLS	49456	9.08	.454	75	5.23	.00039
RLS–StatDegree	44588	6.60	.294	86	4.47	.00038

Table 2.2: Results summary with the medians of the empirical steps distribution, the average time per iteration and the total CPU time to reach a solution when it is reached in all tests. (*) The algorithm is not always able to find the maximum clique; the percent of successes is reported in these cases.

pletely random counterparts. For example ExpPlat–StatDegree finds the maximum clique in $GIL(1100, 0.3)$ only in the 60% of the runs.

Table 2.2 shows that ExpPlat–DynDegree spends less time per iteration (factor of 1.4) than Exp–DynDegree in $PAT(1100, 366)$. The results is confirmed by a Mann-Whitney U-test at significance level 0.05: p-value is $1.593 \cdot 10^{-4}$. The improvement is even bigger in $GIL(1100, 0.3)$ where degree-based selections are less appropriate. In this case the factor is 3.8 and is also confirmed by a Mann-Whitney U-test at significance level 0.05: p-value is $1.649 \cdot 10^{-4}$.

In case of dynamic degree selection, the incremental update routine complexity depends also on the size of the PossibleAdd set. With plateau phases the search is longer and the PossibleAdd set is on average smaller.

RLS, which has a different and less frequent restart policy, alternates between short expansions and plateaus. Therefore the PossibleAdd set remains on average smaller than in Exp–DynDegree or ExpPlat–DynDegree and the cost per iteration is smaller.

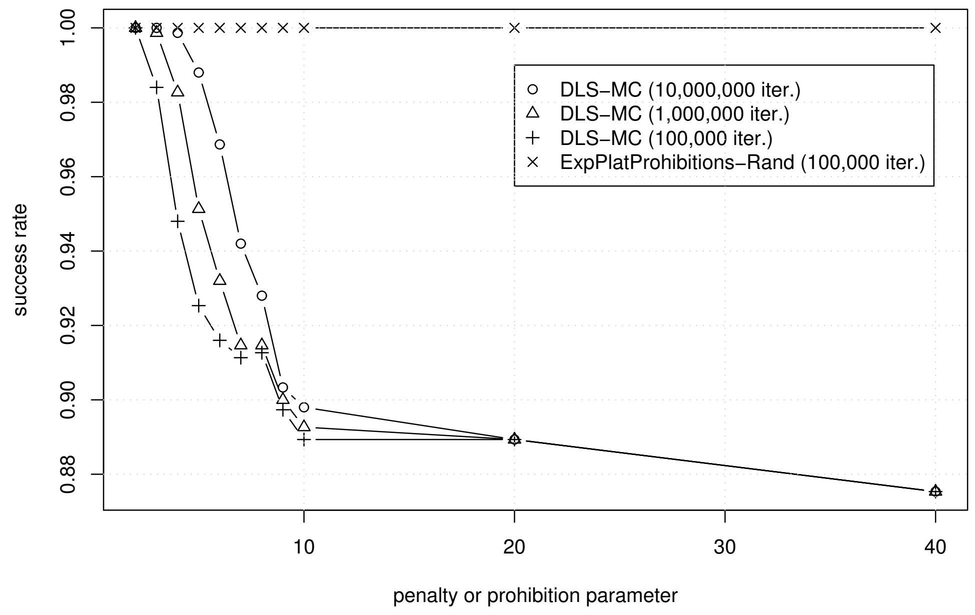


Figure 2.5: Success ratio of penalty- and prohibition-based algorithms on instances of the Preferential Attachment model.

2.3.3 Penalties Versus Prohibitions

As shown in Table 2.2, DLS–MC is not always able to find the best clique on *PAT* graphs while prohibition-based heuristic is always successful. Our results confirm that the penalty heuristic tends to be less robust than the prohibition-based heuristic. A significant dependency between DLS–MC performance and the choice of the penalty delay parameter is also discussed in [54]. Further investigations, summarised in Figure 2.5, show the success rate of DLS–MC compared with that of ExpPlatProhibition–Rand for different values of the penalty delay and prohibition time parameters. The tests are on all instances of the *PAT* graphs of our data set.

ExpPlatProhibition–Rand is always able to find the maximum clique within 100,000 iterations, while DLS–MC fails for several penalty delay values even incrementing the maximum number of iterations by a factor of 10 or 100.

2.4 Conclusions

The results of the tests on the two graph classes show clearly that the plateau search is necessary to find the maximum clique in hard instances and in any case to reduce the average number of iterations. The complexity added to the algorithms by the plateau search does not increase the cost per iteration. On the contrary, especially for the algorithms using the dynamic degree for candidate selections, it reduces the CPU time per iteration.

On Gilbert's graphs, where the nodes have the same degree on average, prohibition- or penalty-based algorithms perform better than pure random selections. On instances of the Preferential Attachment model, algorithms selecting the nodes using information about the degree are faster.

On the contrary, degree-based algorithms have poorer performance than random-selection algorithms in Gilbert's graphs, while prohibition- and penalty-based algorithms are disadvantageous in the Preferential Attachment model. The penalty heuristic is less robust than the prohibition heuristic, depending on the appropriate selection of the penalty value.

RLS and RLS–StatDegree, always perform better then the other algorithms. The cost per iteration of RLS–StatDegree is bigger than the one of DLS–MC, although of the same order of magnitude. But the fewer steps needed on average to find the best cliques make RLS the best choice for the two graph models considered in this chapter, especially considering that no detailed tuning is executed in RLS before running the comparison.

Chapter 3

Search Landscape Visualisation

As a useful complement to the analytical work described in Chapter 2, we propose a set of techniques for the visualisation of search landscapes [45] aimed at supporting the researcher's intuition on the behaviour of a SLS algorithm applied to a combinatorial optimisation problem. We discuss the scalability issues posed to visualisation by the size of the problems and by the number of potential solutions, and we propose approximate techniques to overcome them. The proposed visualisation technique is also capable to rendering explicitly the geographic metaphors used by researchers to describe areas of interest of the landscape, and has therefore a tutorial valence.

3.1 Introduction

Optimisation problems are often characterised by a large number of variables. The set of admissible values for such variables is called *search space*, and can usually be provided with a rich topological structure, which is determined both by the problem's intrinsic structure and by the solving algorithm's characteristics.

A search space complemented with the topological structure induced by the local search algorithm (evaluation function and neighbourhood relation) is called a *search landscape*, and its structure determines, by definition, the behaviour of the solving technique. *Search landscape analysis* is a research field aimed at providing tools for the prediction of the search algorithm's performance and its consequent improvement. Relevant features in this kind of analysis are, of course, the search space size and the number of degrees of freedom (i.e., the dimensionality).

In this chapter, we will focus on Stochastic Local Search (SLS) techniques [38], where a neighbourhood operator is defined in order to map a configuration into a set of neighbouring ones; the relevant topological structure is defined by the chosen neighbourhood operator. An important feature influencing the behaviour of SLS algorithms is the relative position and reachability of local optima with respect to the neighbourhood topology, and some problem instances are known to be hard with respect to SLS algorithms precisely because of 'misleading' sets of good configurations. Researchers often resort to landscape metaphors such as peaks, valleys, plateaux and canyons to describe the possible pitfalls of the techniques, but the sheer dimensionality of the search space often defeats intuition.

We propose a tool for visual analysis of search landscapes for supporting the researcher's intuition via a careful selection of features to be maintained while the space dimensionality is reduced to a convenient size (2 or 3) for displaying. visualisation techniques for complex spaces usually suffer from a number of problems. In particular, the size of the search space requires the display technique to be highly scalable; moreover, the reduction method must be consistent in time, so that subsequent optimisation steps correspond to small variations in the visualised landscape: such continuity is necessary to help the researcher consider the optimisation process as a whole, rather than focus on single snapshots.

The Maximum Clique problem will be used as a paradigmatic example, together with two state of the art SLS algorithms for its optimisation. By means of such examples, the scalability and continuity issues presented above will be discussed and tested, and the behaviour of the solving techniques on hard instances will be analysed.

The remainder of the chapter is structured as follows. In Section 3.2 a brief overview of previous relevant work on visualisation for optimisation algorithms is presented. In Section 3.3 a representation of the search landscape in three dimension is proposed. In Section 3.4 an approximated representation is proposed, which scales better with the dimensions of the search landscapes, because it does not require the enumeration of the exponential number of sub-optimal solutions. In Section 3.5 a case study is presented, which shows how the behaviour of a penalty based algorithm can be analysed from the changes in the landscape after the penalisation phases. Finally, in Section 3.6 conclusions are drawn and some ideas for further developing and leveraging this new type of analysis are presented.

3.2 Previous and Related Work

The last years have witnessed a boost in the research on complex systems visualisation, due to the general availability of inexpensive hardware for the fast computation of linear transformations involved in 3D polygon display. Graphics Processing Units (GPUs) are available in all display cards, and specialised expansions with hundreds to thousands of GPUs are available for common bus architectures for co-processing purposes. Therefore, the main scalability issues connected to large set visualisation can be overcome by brute force. Work on dimensionality reduction via sampling with preservation of relevant graph properties has been presented in [56]. Online drawing of graphs is studied, for instance, in [30], where the effort is focused at preserving the presentation layout while the graph is changing.

Work that combines visualisation and optimisation dates back to [51], where multidimensional scaling and other techniques are applied to the visualisation of evolutionary algorithms, while other contributions are aimed at human-guided search [1] where the computer finds local optima by hill-climbing while the user identifies promising regions of the search space. visualisation of Pareto-Sets in Evolutionary Multi-Objective optimisation is investigated in [42] by finding a mapping which maintains most of the dominance relationships. In [36] the authors propose a visualisation suite for designing and tuning SLS algorithms. Starting from a selection of candidate solutions, the visualisation tool uses a spring-based layout scheme to represent the solutions in two dimensions. The algorithm execution traces are then represented as trajectories around the laid out solutions, and the resulting analysis used by the researchers to tune the algorithm studied. While in [36] the authors focus on the behaviour of the optimisation algorithm and on the human interaction for the tuning of the algorithm parameters, we are interested in analysing the intrinsic properties of the problem instance.

3.3 Complete Three-Dimensional Landscapes

The number of dimensions in the search space of a combinatorial optimisation problem is equal to the number of variables in the instance. We propose a new way to represent it in a lower-dimensional space, describing the landscape as the variation of the solution quality (i.e., objective function) or the variation of a heuristic guidance function for the specific SLS algorithm (i.e., evaluation function).

For example, in order to reduce the problem landscape from the n dimensions to just 2, one could represent the feasible solutions as points plotted against their quality. In such two-dimensional plot, however, the ordering of the points representing the solutions is arbitrary, as it happens for example with plateau connection graphs and barrier-level basin graphs (see [38] for a thorough review of analysis methods).

A good layout for the solutions tries to preserve the distance (similarity) among the solutions in the original n-dimensional space. Our aim is to find a representation which can be easily visualised, therefore we will concentrate on reductions of the search space to a three-dimensional landscape, which also allows for some intuitive representation of the possible basic steps of a SLS algorithm.

In the following, we map the objective function value to the z axis, so that the z quota of a solution will always be fixed. The aim of the proposed techniques is to find convenient x and y coordinates of each solution. Since in our experience natural landscapes are in three dimensions, also the metaphor of *landscape* and of evaluation function can be easily represented and understood in three dimensions. It allows for a natural visualisation of valleys, plateaus, and peaks, and fits perfectly with the basic operations of SLS algorithms.

3.3.1 The Technique

The first possible approach consists of computing a 3D layout of the search landscape of the problem instance. Figure 3.1 shows the landscape of an instance of the MC problem represented as a neighbourhood graph in three dimensions, where nodes correspond to feasible solutions (cliques) and edges correspond to the neighbourhood structure (i.e., cliques of Hamming distance 1 or 2). Node heights represent the size of the corresponding clique, and the horizontal layout tries to retain the topology of the neighbourhood graph.

The lowest vertices represent cliques with the smallest size (2). In all levels the number of vertices depends on the connectivity in the instance

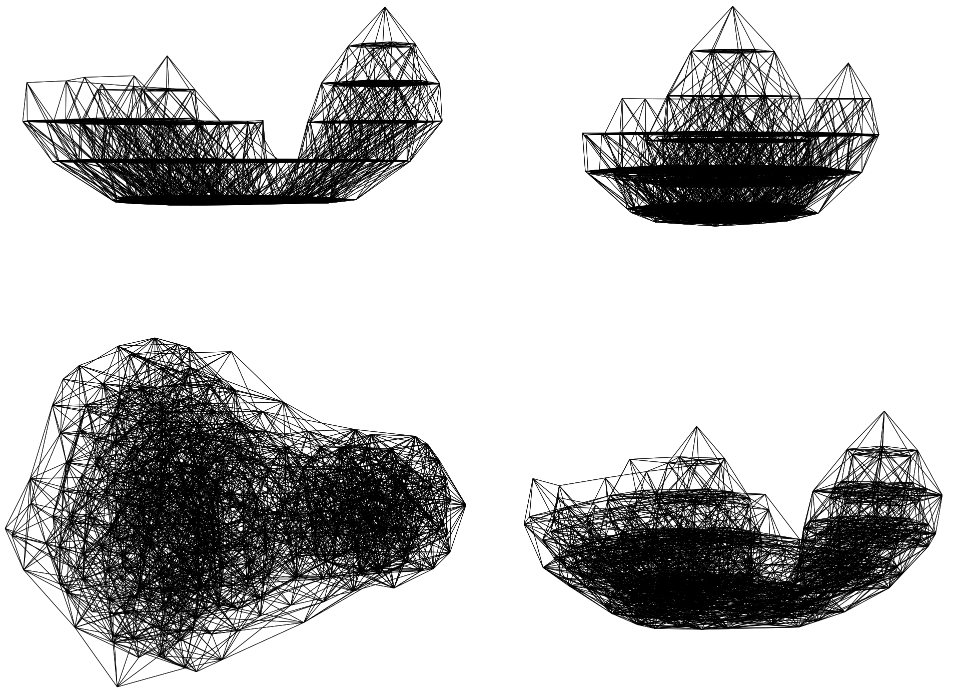


Figure 3.1: Search Landscape corresponding to a Brockington-Culberson graph with 20 nodes, edge density 0.5, and maximum clique of size 7. The four subfigures are four different perspectives on the same 3D model. Brockington-Culberson graphs are designed with the aim of hiding the maximum clique [16].

graph, and in the instance depicted in Figure 3.1 are bounded from above by $\binom{20}{k}$ where k is the size of the cliques in the level, and 20 the number of nodes in the specific instance.

The landscape depicted in Figure 3.1 is generated starting from 43 maximal cliques enumerated empirically with two state of the art heuristic algorithms for the MC problem: RLS–MC and DLS–MC. It has to be noted that a complete enumeration should always be used when possible, because using SLS algorithms for the empirical enumeration could lead to a bias in the representation. From every maximal clique, a tree containing all possible solutions within the maximal one is generated by means of a backtracking algorithm. Solution trees originated from different maximal cliques can overlap and share consistent parts of the search space. Therefore, during the enumeration all the solutions are added to a hash table, and when a solution is encountered twice the corresponding sub-tree is pruned. Once all the solution are enumerated, they are connected with arcs if a local search algorithm could move from one to the other by means of one of the neighbourhood operations (*add*, *drop*, or *swap*). Once the graph is constructed a spring-based method is used to lay out the graph (with the further constraint that all vertices lay on the plane corresponding to their objective value). The nodes are treated as point-wise unit masses subject to pairwise forces of two types. The first is an attractive spring force based on a smoother version of Hooke's law [26]; it accounts for the node adjacency. The force acting on node a due to node b is

$$\boldsymbol{F}^H_{ab} = \begin{cases} k_{ab} \dfrac{\boldsymbol{b}-\boldsymbol{a}}{\|\boldsymbol{b}-\boldsymbol{a}\|_2} \log \dfrac{\|\boldsymbol{b}-\boldsymbol{a}\|_2}{r_{ab}} & \text{if } a \text{ and } b \text{ are adjacent} \\ 0 & \text{otherwise,} \end{cases} \tag{3.1}$$

where $\boldsymbol{a}$ and $\boldsymbol{b}$ are the coordinate vectors of a and b in the low dimensional representation, r_{ab} is the ideal distance and k_{ab} is the spring stiffness, which depends on the desired layout. The second force is of Coulombian type and acts between every pair of nodes:

$$\boldsymbol{F}^{C}_{ab} = \frac{q_a q_b}{\|\boldsymbol{b}-\boldsymbol{a}\|_2^{D_{ab}}} \frac{\boldsymbol{a}-\boldsymbol{b}}{\|\boldsymbol{b}-\boldsymbol{a}\|_2}, \tag{3.2}$$

where $q_a = q_b$ and the exponent D_{ab} depends on a threshold distance r_{th}:

$$D_{ab} = \begin{cases} 2 & \text{if } \|\boldsymbol{b}-\boldsymbol{a}\|_2 \leq r_{\text{th}} \\ 4 & \text{otherwise.} \end{cases}$$

The snapshots in Figure 3.1 show the landscape from the side, the front, the top, and in perspective. The almost flat area corresponding to a plateau bumped with several local optima is quite evident, as well as the clique of size 7 and the barrier between the points on the plateau and the maximum clique. This provides immediate information about the instance properties: because of the low number of nodes shared by the optimum and the flat area, algorithms with long plateau phases could be worse than algorithms with shorter plateau phases and more frequent restart policies. The layout can also embed extra information. For example, the vertices can be rendered with different colours depending on how frequently they are visited by the SLS algorithm, in order to analyse the attraction basins and how they are distributed with respect to the global optimum. Another example of information that could be easily embedded by means of a vertex colouring is the average degree distribution of the nodes in the solutions, which can give an immediate summary of the degree distribution and give some hints on the performance of greedy local search algorithms.

3.3.2 The Tool

The described technique works with all combinatorial optimisation problems. Due to the way the application currently enumerates the possible solutions, the visualisation is limited to the problems whose solutions can be encoded with binary strings and all sub-strings are also feasible solutions.

Figure 3.1, as well as all the following figures are produced with *Graph Visualizer*, an application for Mac OS X that has been developed with the specific purpose of helping the researchers' intuition when studying the landscapes of combinatorial optimisation problems[1]. Graph Visualizer is a general purpose tool for laying out graph structured data. Custom-designed tools for specific analysis purposes can be plugged into the main program; in particular, the Search Landscape Visualisation (SLV) plugin produces three dimensional landscapes starting from a set of maximal solutions. The landscape generation is completely automated, the only input required is the list of local optima for the given instance, which we assume is automatically generated by the optimisation program. Both the main multi-threaded application and the SLV plugin have been developed in Objective-C using the OpenGL libraries for the 3D visualisation and Cocoa libraries for the native Mac OS X user interface. The whole application including the layout algorithm has been written from scratch by the authors.

3.3.3 NURBS Covers

For easier visualisation, the search landscape can be covered with Non-uniform rational B-spline (NURBS) surfaces. These surfaces are superimposed over the three-dimensional landscapes by setting the height of the surface control points to the same heights of the corresponding vertices of the three dimensional neighbourhood graph.

The NURBS surface has degree 3 and is controlled by a user-defined number of evenly spaced control points. The more the control points the more precise the representation, but too many control points can lead to artificial local optima between the solutions where control point heights are not set by any solution.

[1]The software is available for research purposes at `http://graphvisualizer.org/`

The colouring of the NURBS surfaces as well as the clusters in the approximate landscapes varies from blue to red showing the quality of the solutions.[2]

3.4 Approximated Landscapes

While the search space analysis of small instances is interesting *per se*, every solution of size m contains $\binom{m}{k}$ solutions of size k, and the enumeration of all possible solutions at the lower levels (avoiding repetitions) does not scale with solution size. In order to handle larger instances, a number of approximated layouts can be introduced. The first solution considers clusters of sub-cliques as a unique object, the second operates by subsampling the solutions obtained by the SLS algorithm.

3.4.1 Clusters of Solutions

The following technique for generating an approximate landscape does not require the enumeration of all sub-optimal solutions, but just clusters of solutions having mutual Hamming distance 2.

Starting from local optimum solutions of size m, the cluster of solutions of size $m - k$ will contain $\binom{m}{m-k}$ solutions having mutual Hamming distance 2. The clusters can be scaled properly depending on the number of solutions they contain and the whole tree structure rooted in the local optimum is reduced to a stack of clusters with different sizes. Of course clusters belonging to stacks below different optimal solutions overlap for a volume which is proportional with the number of common solutions.

Let C_1 and C_2 be two maximal solutions; let $m = |C_1|$ be the size of the first, and $s = |C_1 \cap C_2|$ be the number of common components. The fraction of the cluster of solutions of size k below C_1 overlapping with

[2]Examples of coloured surfaces are available at `http://graphvisualizer.org/slv`.

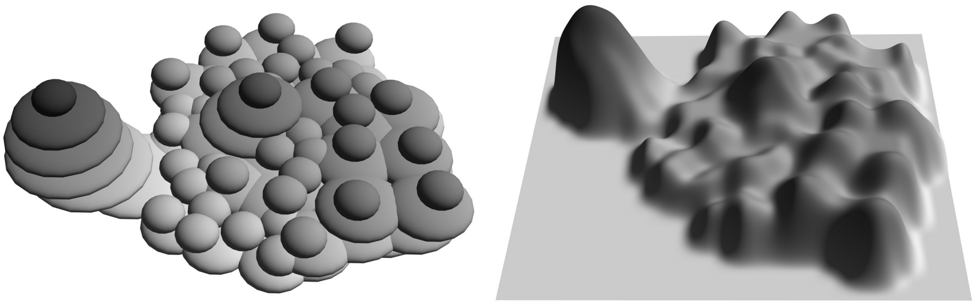


Figure 3.2: On the left an approximated Search Landscape corresponding to a Brockington-Culberson graph with 20 nodes, edge density 0.5, and maximum clique of size 7. The approximated landscape retains the same shape as the complete landscape in Figure 3.1. On the right the approximated landscape is covered with a NURBS surface with 30×30 evenly spaced control points.

the corresponding cluster of solutions of the same quality below C_2 is the following:

$$\binom{m}{k}\binom{s}{k}^{-1} = \frac{m!(s-k)!}{s!(m-k)!}. \tag{3.3}$$

With this technique, there is no need to enumerate the exponentially large number of sub-optimal solutions: knowing the size of the clusters and the fraction of their volume that overlaps is enough to render an approximated landscape like the one shown in Figure 3.2.

The multidimensional scaling is done with the spring based layout technique used in Section 3.3, but this time the vertices to be laid out are the clusters, their size is reflected in the charges q_a and q_b that determine their repulsive force in (3.2), and their overlapping volumes are encoded in the spring elastic constants K_{ab} and their zero-energy spring lengths r_{ab} of (3.1).

The computation of large binomial coefficients is performed by the Stirling approximation of the factorials:

$$\ln n! \approx (n + 0.5)\ln n - n + \frac{\ln(2\pi)}{2}$$

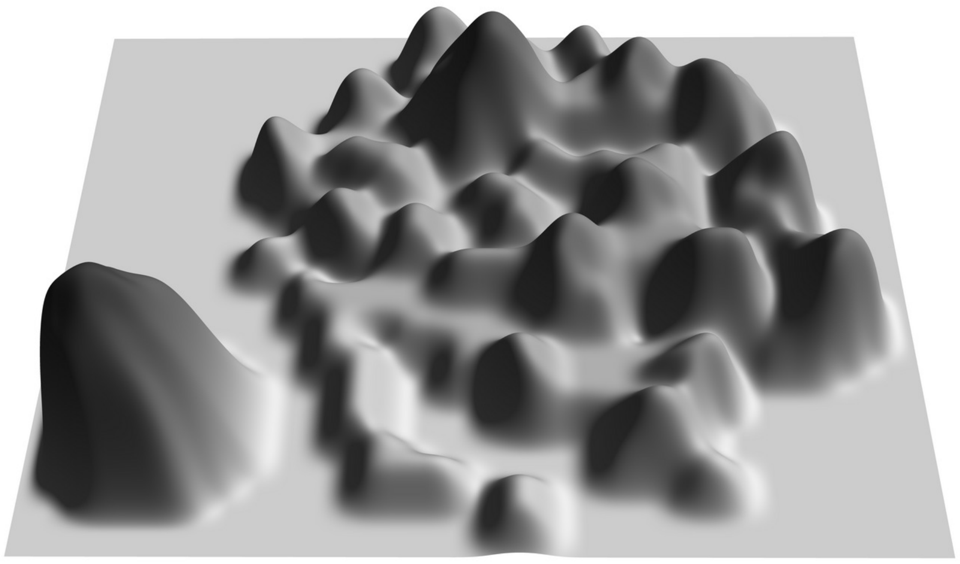


Figure 3.3: Same Landscape of Figure 3.2 but subsampling the search space removing solutions with quality less than 3. This highlights the barrier between the plateau and the optimum.

therefore (3.3) can be approximated by

$$\binom{m}{k}\binom{s}{k}^{-1} \approx e^{(m+0.5)\ln m+(s-k+0.5)\ln(s-k)-(s+0.5)\ln s-(m-k+0.5)\ln(m-k)} \tag{3.4}$$

and it can be computed for large values of m and s.

3.4.2 Search Space Sampling

We can consider a reduction in the number of solutions around the global optimum, by filtering out the solutions which share fewer components with the global optimum. Another possible sampling strategy is to reduce the depth of the trees rooted in the local optima. The SLV plugin supports the combination of the two strategies. The sampled portion of the search landscape can improve the visualisation by enhancing some of its features, but it can also drastically change the properties of the landscape. For example, Figure 3.3 shows the same landscape of Figure 3.2, obtained by applying the approximation technique in Section 3.4.1 and restricting the solutions to be considered to the ones having a quality (size) of at least 3. The restriction on the quality of the solutions to be considered makes the barrier more evident, but it also makes the landscape disconnected.

3.5 Dynamic Landscapes

In the following section, we will show through an example how the proposed analysis of the Search Landscape sheds some light on the dynamics of penalty-based SLS algorithms, and on the changes of the evaluation function g during the search.

When DLS–MC reaches a local optimum, all the components belonging to such solution are penalised. The aim of the penalisation is to drop the quality of the local optimum and render it less attractive in the subsequent steps of local search. Nevertheless, the penalisation effect is not limited to the local optimum, but impacts all the areas of the landscape having solutions overlapping with the penalised one. Therefore, it is of particular interest to study the behaviour of the penalisation and its impact on the landscape.

The adopted technique is composed of two steps. First, the three dimensional landscape corresponding to the objective function f is laid out by means of the force based multidimensional scaling technique described in Section 3.3. Then a landscape corresponding to the evaluation function g for each penalisation step is produced. For the continuity reasons stated in Section 3.1, the horizontal layout of the objective function search landscape is retained throughout all penalisation steps, the only thing that varies is the quality of the solutions whose components are penalised.

Figure 3.4 shows the penalisation effect which transforms the landscape of the Brockington-Culberson instance of Figure 3.1. In Figure 3.4 the first NURBS surface is produced from the complete representation of the objec-

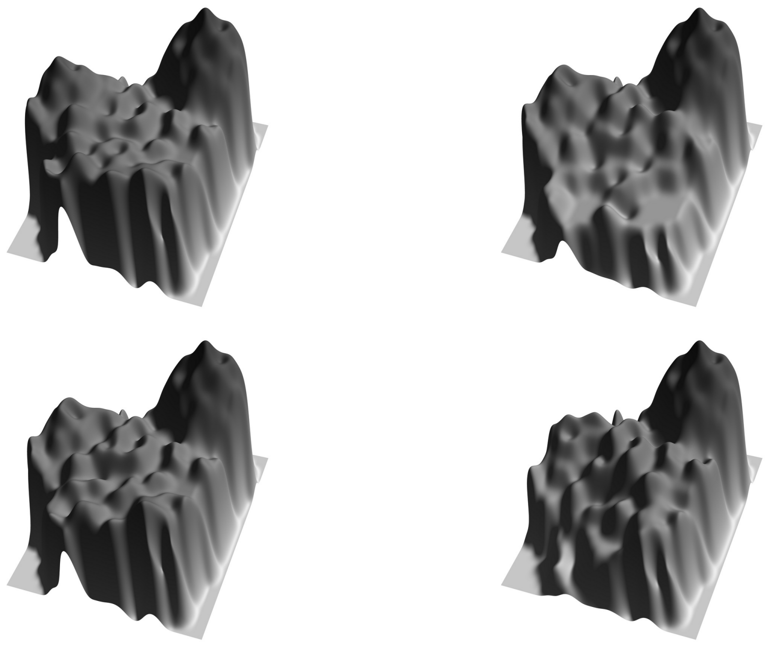


Figure 3.4: Four penalisation steps of DLS–MC on the Brockington-Culberson instance of Figure 3.1. The steps are shown chronologically from left to right and from top to bottom.

tive function. The second landscape in figure retains the same horizontal layout, and the plateau is partly flattened by the penalisation effect, which is then partially reverted after the penalties expiration in the third landscape. The fourth landscape shows the last penalisation before DLS–MC is able to find the optimum solution. The effect is more clear in Figure 3.5, in which the plateau size reduction is quite evident. The increased number of levels in the graph after the penalisation is due to the fact that the landscape corresponds to an evaluation function g and not an objective function. The algorithm whose steps are shown in figure associates integer penalties to the solution components belonging to local optima. The evaluation function g is computed as the cardinality of the solution minus the penalties associated to its components, therefore the landscape is on discrete levels, some of which have a negative quality.

The penalisation strategy was effective in finding the well hidden global

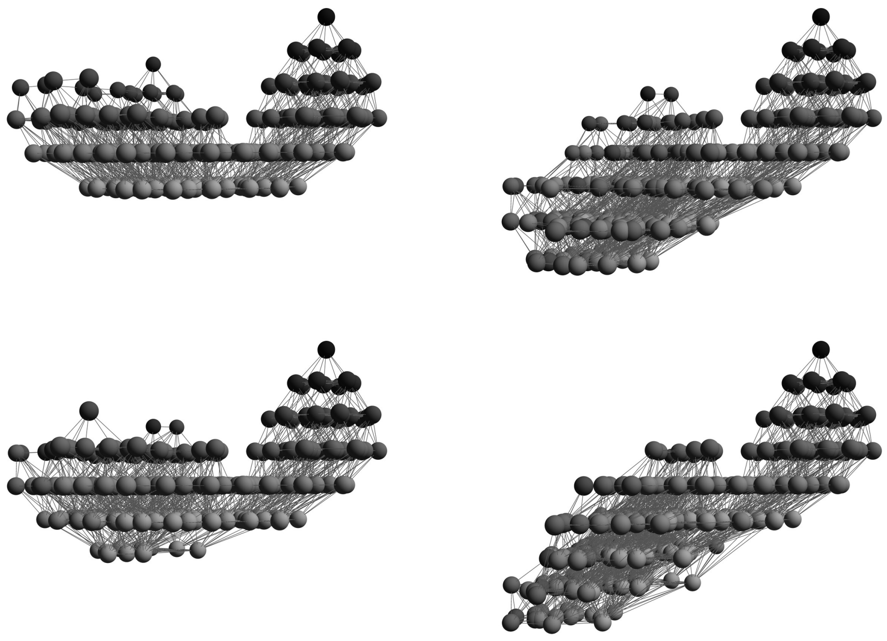


Figure 3.5: Four penalisation steps of DLS–MC on the Brockington-Culberson instance of Figure 3.1. The steps are shown chronologically from left to right and from top to bottom.

optimum, which does not share solution components with the penalised local optima.

On the contrary, in the instance of the MC problem depicted in Figure 3.6 the maximum clique has size 5, and the 30 smaller cliques of size 4 share a node with the maximum one. Therefore a penalisation of a local maximum always impacts the global one.

Figure 3.7 represents the results of a DLS–MC run on the instance described above. The NURBS landscape on the top-left represents the unmodified objective function, and it shows in the middle the global optimum slightly above the other optima. The other three landscapes in figure show the evaluation function after subsequent penalisation steps. The penalisation always impacts on the global optimum.

In order to highlight the effect without using colours, the quality of the solutions in Figure 3.7 has been emphasised.

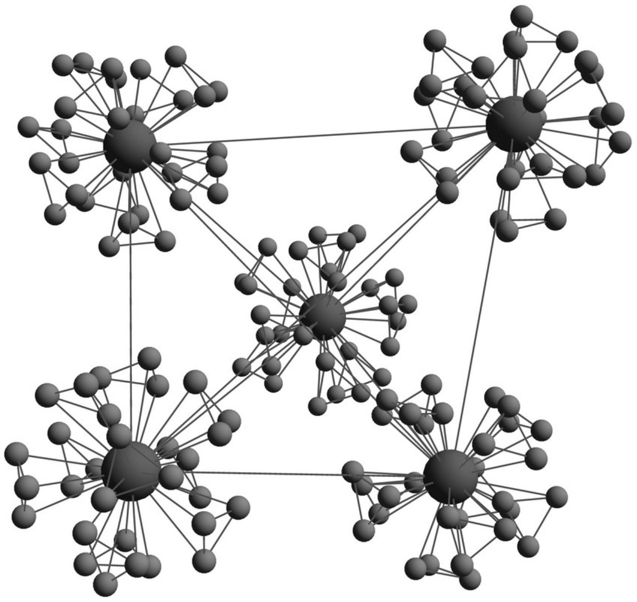


Figure 3.6: A MC instance with 155 nodes. The maximum clique has size 5, and the 30 smaller cliques of size 4 share a node with the maximum one.

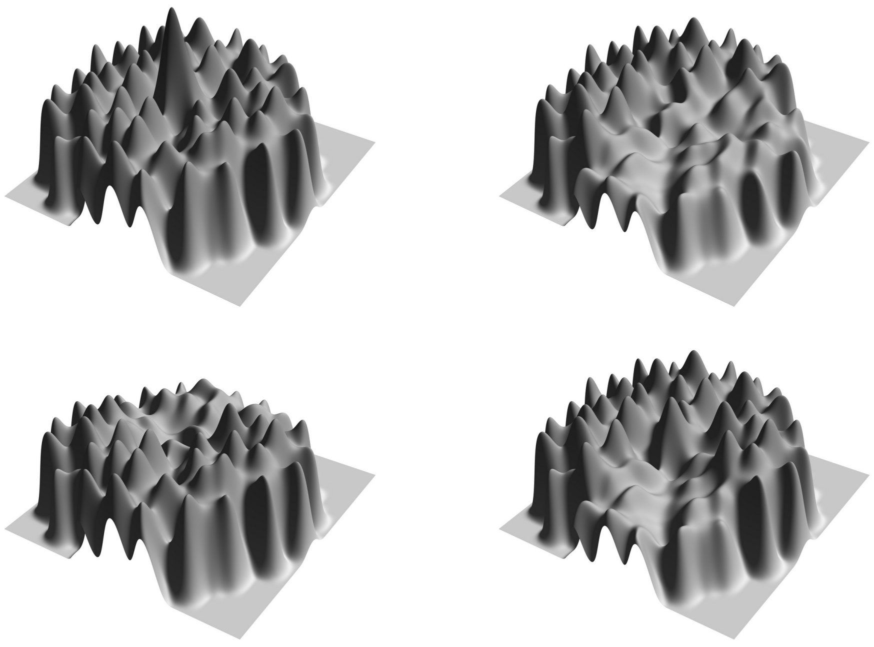


Figure 3.7: Four penalisation steps of DLS–MC on the instance depicted in Figure 3.6. The first landscape on the top-left shows the objective function with the global optimum in the middle.

3.6 Conclusions

We have presented a set of techniques for the visualisation of search landscapes which can support the researcher's intuition on the behaviour of a SLS algorithm applied to combinatorial optimisation problems. The visualisation also renders explicitly the geographic metaphors used by researchers to describe areas of interest of the landscape.

The examples presented in this chapter are small instances useful to show how some features of the landscapes are rendered with the proposed techniques. The approximation techniques presented in Section 3.4 allow for the representation of instances otherwise intractable for the complete representation, while maintaining the features of the complete enumeration.

Current research is aimed towards more scalable layout algorithms with no exogenous parameters that can lay out landscapes with more than few thousand solutions and tens of thousands of relations among them. The convergence of a non-hierarchical spring-based layout algorithm depends strongly on the user provided parameters like the repulsion force, damping factor, zero energy spring lengths, spring elastic constants, which have to be appropriate for the graph structure.

The proposed techniques have been implemented in a Mac OS X application that allows for real-time manipulation and animation. The program is free for academic use and can be downloaded from

`http://graphvisualizer.org/`

Chapter 4

Engineering an Efficient Algorithm

Building on the results of the empirical analysis in Chapter 2 and from the profiling of various algorithm implementations, in this chapter we propose a new algorithm that presents two kinds of changes with respect to the original RLS version. The first changes are algorithmic and influence the search trajectory, while the second one refers only to the more efficient implementation of the supporting data structures, with no effect on the dynamics.

4.1 Introduction

The algorithmic changes suggested by prior analysis are the following ones. In the previous version, the search history was cleared at each restart and the tabu tenure reset to $\mathsf{MIN\_T} = 1.1$. Now, in order to allow for a more efficient diversification, the entire search history is kept in memory and the tabu tenure is never reset. To underline this fact, the new version is called RLS 'Long Term Memory', or RLS–LTM.

Having a longer memory causes the parameter T to explode on some specific instances characterised by many repeated configuration during the search. If the prohibition becomes much larger than the current clique size, after a maximal clique is encountered and one node has to be extracted from the clique, all other nodes will be forced to leave the clique before the first node is allowed to enter again. This may cause spurious oscillations in the clique membership which may prevent discovering the globally optimal clique. An effective way to avoid the above problem is to put an upper-bound $\mathsf{MAX\_T}$ equal to the current estimate of the maximum clique, i.e., $\mathsf{MAX\_T} = |\mathsf{Best}| + 0.5$, where $|\mathsf{Best}|$ is the size of the best clique found so far.

4.2 Implementation Details and Cost per Iteration

The total computational cost for solving a problem is of course the product of the number of iterations times the cost of each iteration. More complex algorithms like RLS risk that the higher cost per iteration is not compensated by a sufficient reduction in the total number of iterations. This section is dedicated to exploring this issue.

The original implementation [13] focused on the algorithm and the appropriate data structures but did not optimise low-level implementation details. For example, every time a new configuration had to be inserted in the table, the memory needed to store the element was allocated dynamically. The new implementation moved from these on-demand allocations to the more efficient allocation of a single bigger chunk of memory; the memory is used as a pool of available locations to be assigned to the elements when needed.

Moreover, the hash table containing the configurations resolves key collisions by means of chaining. In order to keep the frequent access operation as close to a direct access as possible, these chains have to be kept as short as possible. This has been achieved by doubling the size of the hash table when the number of elements inside the table exceeds a specified load factor. In this way the amortised complexity of the access operations is kept

Instance	Steps per second		Speedup
	RLS [13]	RLS–LTM	
gilbert_1100_0.3	11202	107527	9.6
pa_1100_366	24839	168350	6.8
C125.9	371747	1162791	3.1
C250.9	281690	943396	3.3
C500.9	165289	714286	4.3
C1000.9	80451	471698	5.9
C2000.9	27285	265957	9.7
DSJC500_5	43290	295858	6.8
DSJC1000_5	17422	160000	9.2
C2000.5	5573	78125	14.0
C4000.5	1537	34965	22.8
MANN_a27	485437	909091	1.9
MANN_a45	293255	425532	1.5
MANN_a81	14286	16667	1.2
brock200_2	109769	543478	5.0
brock200_4	147493	699301	4.7
brock400_2	103413	555556	5.4
brock400_4	105374	552486	5.2
brock800_2	33715	264550	7.8
brock800_4	33311	262467	7.9
gen200_p0.9_44	321543	1000000	3.1
gen200_p0.9_55	273224	943396	3.5
gen400_p0.9_55	210084	800000	3.8
gen400_p0.9_65	204499	740741	3.6
gen400_p0.9_75	205761	724638	3.5
hamming10-4	46339	316456	6.8
hamming8-4	113122	568182	5.0
keller4	140647	546448	3.9
keller5	55036	296736	5.4
keller6	7011	101626	14.5
p_hat300-1	57870	308642	5.3
p_hat300-2	112233	558659	5.0
p_hat300-3	171821	729927	4.2
p_hat700-1	21758	168350	7.7
p_hat700-2	49358	337838	6.8
p_hat700-3	88417	478469	5.4
p_hat1500-1	7345	85470	11.6
p_hat1500-2	13504	184843	13.7
p_hat1500-3	30460	282486	9.3

Table 4.1: Speed improvement on random graphs and selected DIMACS benchmark instances of the new RLS implementation. Some round figures are due to the internal clock resolution.

close to O(1).

The speedup results are reported in Table 4.1. They show the improvement in the steps per seconds achieved by the new version, for two random graphs and some representative DIMACS instances. The number of steps per second is computed by measuring on every instance the CPU time spent by the two algorithms to perform 1000000 iterations. Let us note that the obtained speedup is substantial. For example, the improvement for large random graphs increases with the graph dimension reaching a factor of 22 for graphs with thousand nodes (C4000.5).

Let us now consider a simple model to capture the time spent by the RLS algorithm on each iteration. Most the cost is spent on updating the data structures after each addition or deletion. After a node deletion the complexity for updating the data structures is $O(deg_{\overline{G}}(v))$, $deg_{\overline{G}}(v)$ being the degree of the just moved node v in the complementary graph $\overline{G}$. After a node addition the complexity is $O(deg_{\overline{G}}(v) \cdot |\textsc{PossibleAdd}|)$, see [13] for more details. Now, because the algorithm alternates between expansions and plateau moves, for most of the run |PossibleAdd| oscillates between 0 and 1. We can therefore make the strong assumption that |PossibleAdd| is substituted with a small constant. In both cases the dominant factor is therefore $O(deg_{\overline{G}}(v))$.

The computational complexity for using the history data-structure can be amortised to a $O(1)$ complexity per iteration. The restart operation cannot be amortised: its complexity is $O(n)$ but it is not performed regularly. On the contrary, the number of restarts highly depends on the search dynamics and on the hardness of the instance.

Under the above assumption we propose an empirical model for the time per iteration which is linear in the number of node and the degree:

$$T(n, deg_{\overline{G}}) = \alpha\ n + \beta\ deg_{\overline{G}} + \gamma \tag{4.1}$$

The last simplification is given by substituting the average node degree instead of the actual degree.

Let us note that the above model is not precise if the size of the PossibleAdd set remains large for a sizeable fraction of the iterations. For example this is the case when a large graph is extremely dense, and the clique is very large. In this case the size of the PossibleAdd set is a non-negligible factor which multiplies $deg_{\overline{G}}(v)$, impacting significantly the overall algorithm performance. This happens for the MANN instances in the DIMACS benchmark set which are not considered when fitting the above model.

We fit the model on our reference machine, having one Xeon processor at 3.4 GHz and 6 GB RAM. The operating system is a Debian GNU/Linux 3.0 with kernel 2.6.15-26-686-smp. The algorithm is compiled with g++ compiler with '-O3 -mcpu=pentium4'. The fitted model is the following one:

$$T(n, deg_{\overline{G}}) = 0.0010\ n + 0.0107\ deg_{\overline{G}} + 0.0494 \tag{4.2}$$

The fit residual standard errors for α, β and γ are 0.0004, 0.0009 and 0.2765 respectively.

Let us note that the cost for using the history data-structure, which is approximately included in the constant term in the above expression, becomes rapidly negligible as soon as the graph dimension and density are not very small. In fact the memory access costs approximately less than 50 nanoseconds per iteration while the total cost reaches rapidly tens of microseconds in the above instances.

To assess the scalability of RLS–LTM we show in Figure 4.1 the average CPU time per iteration on Gilbert's graphs of different sizes. The class of graph instances as well as the algorithms depicted are described in Chapter 2. The regression lines (in log-log scale) have a slope of 2.41, 0.92 and 0.95 for EXP–DYNDEGREE RLS–LTM and DLS–MC(PD=2) respectively, confirming an approximate cost per iteration of EXP–DYNDEGREE growing faster than n^2, while RLS cost, even if the candidate selection is based on the dynamic degree, grows approximately linearly.

4.2.1 DIMACS Benchmark Set

Table 4.2 compares DLS–MC(PD=OPT) and RLS–LTM on a selected 'snapshot' of the DIMACS benchmark set. The results presented are aver-

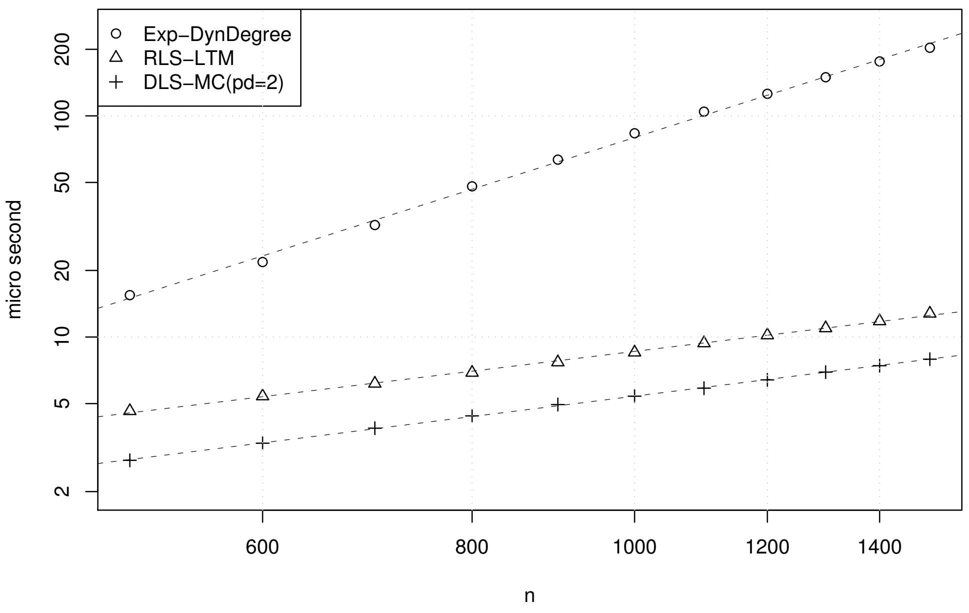


Figure 4.1: Empirical cost per iteration in μseconds on Gilbert's graphs. Log-log scale.

ages on 100 runs of 100,000,000 maximum steps each. DLS–MC(PD=OPT) is DLS–MC with the *pd* parameter set to the optimal value for each single instance as suggested in [54].

For each instance and both algorithms the table shows the median clique size with the median percentage deviation from the best known. The CPU time and iteration medians and interquartile ranges (IQRs) are reported only for successful runs.

One algorithm dominates the other if it has a bigger median solution quality, or a smaller median percentage deviation from the best known. If both algorithms are able to find the maximum clique on every run, the dominating algorithm is the one having either a smaller median CPU time or, in the case of no measurable difference in the CPU times, a smaller number of iterations. The comparisons are assessed statistically by means of a Mann-Whitney U-test (Wilcoxon rank-sum test) at significance level 0.05. The dominating algorithm is highlighted in bold.

Let us note again that the comparison below is not fair, because in one case (DLS–MC) one reports only the time corresponding to the optimal setting of an individual *pd* parameter for each instance, while in the second case this extensive tuning phase is absent.

In most cases, apart from the 'camouflaged' Brockington-Culberson graphs [16], the optimal values obtained are the same. These graphs have been designed to be difficult for greedy algorithms, therefore it is not surprising that the greedy node selection of RLS negatively impacts on the performance on those graphs. For the CPU times, in many cases the graph dimension is so small that the measure becomes difficult, in some other cases RLS has times which are larger but of the same order of magnitude. For other instances RLS CPU time is shorter, which is quite unexpected given the absence of the tuning phase.

4.3 Conclusions

The results of the investigation show that a careful implementation of the data-structures considering also operating system services like memory allocation achieves a significant reduction of the CPU time per iteration. The implementation of the supporting data structures of the new version has many improvements: i) the management of the dynamic memory, used for storing the configuration fingerprints in the history, which is not allocated when needed as in the first version, but rather pre-initialised and shared among the steps of the actual run; ii) the usage of a dynamic hash table where the size is adapted to the load factor; iii) the substitution of all dynamic allocations in the functions with allocations executed at the

Instance	Best	DLS–MC (pd=opt)			RLS–LTM		
		Solution quality	CPU(s)	Steps	Solution quality	CPU(s)	Steps
C125.9	34	34 (0.00)	< 0.001	175 (215)	**34 (0.00)**	< 0.001	**88 (118)**
C250.9	44	44 (0.00)	< 0.001	1 348 (1 770)	44 (0.00)	< 0.001	1 060 (1 010)
C500.9	57	**57 (0.00)**	**0.030 (0.040)**	**59 780 (76 900)**	57 (0.00)	0.115 (0.253)	82 740 (181 010)
C1000.9	68	**68 (0.00)**	**0.360 (0.630)**	**409 500 (716 300)**	68 (0.00)	1.465 (2.035)	703 000 (973 400)
C2000.9	78	78 (0.00)	-	-	78 (0.00)	-	-
DSJC1000_5	15	15 (0.00)	0.330 (0.527)	81 560 (130 700)	**15 (0.00)**	**0.200 (0.335)**	**31 720 (53 220)**
DSJC500_5	13	13 (0.00)	0.010 (0.010)	2 454 (3 954)	**13 (0.00)**	**0.000 (0.010)**	**1 131 (1 692)**
C2000.5	16	16 (0.00)	0.370 (0.743)	59 000 (117 180)	**16 (0.00)**	**0.465 (0.640)**	**35 760 (49 150)**
C4000.5	18	18 (0.00)	119.000 (143.920)	9 686 000 (11 710 000)	**18 (0.00)**	**107.000 (147.820)**	**3 705 000 (5 113 000)**
MANN_a27	126	**126 (0.00)**	**0.020 (0.010)**	**60 240 (26 820)**	126 (0.00)	0.070 (0.073)	62 760 (72 060)
MANN_a45	345	344 (0.29)	-	-	344 (0.29)	-	-
MANN_a81	1099	1 098 (0.09)	-	-	1 098 (0.09)	-	-
brock200_2	12	**12 (0.00)**	**0.010 (0.020)**	**12 230 (20 680)**	12 (0.00)	0.090 (0.150)	49 100 (83 670)
brock200_4	17	**17 (0.00)**	**0.030 (0.040)**	**39 960 (51 020)**	17 (0.00)	0.190 (0.493)	135 000 (346 480)
brock400_2	29	**29 (0.00)**	**0.230 (0.362)**	**232 800 (371 300)**	29 (0.00)	-	-
brock400_4	33	**33 (0.00)**	**0.030 (0.040)**	**28 770 (37 720)**	33 (0.00)	3.365 (4.530)	1 890 000 (2 521 300)
brock800_2	24	**24 (0.00)**	**7.870 (10.178)**	**3 397 000 (4 387 000)**	21 (14.29)	-	-
brock800_4	26	**26 (0.00)**	**3.275 (4.442)**	**1 410 000 (1 914 600)**	21 (23.81)	-	-
gen200_p0.9_44	44	44 (0.00)	< 0.001	1 934 (3 981)	44 (0.00)	< 0.001	1 535 (1 832)
gen200_p0.9_55	55	**55 (0.00)**	< 0.001	**333 (827)**	55 (0.00)	< 0.001	596 (562)
gen400_p0.9_55	55	**55 (0.00)**	**0.010 (0.020)**	**33 920 (57 910)**	55 (0.00)	0.030 (0.040)	21 160 (30 150)
gen400_p0.9_65	65	**65 (0.00)**	< 0.001	**1 001 (1 480)**	65 (0.00)	< 0.001	1 294 (1 132)
gen400_p0.9_75	75	**75 (0.00)**	< 0.001	**507 (777)**	75 (0.00)	< 0.001	1 576 (1 184)
hamming8-4	16	16 (0.00)	< 0.001	28 (16)	**16 (0.00)**	< 0.001	**16 (0)**
hamming10-4	40	40 (0.00)	0.000 (0.010)	2 469 (3 241)	**40 (0.00)**	**0.000 (0.010)**	**529 (1 044)**
keller4	11	11 (0.00)	< 0.001	40 (44)	**11 (0.00)**	< 0.001	**11 (8)**
keller5	27	27 (0.00)	0.010 (0.020)	6 345 (8 798)	**27 (0.00)**	**0.010 (0.030)**	**2 828 (7 150)**
keller6	59	59 (0.00)	-	-	**59 (0.00)**	**6.765 (12.482)**	**686 500 (1 265 300)**
p_hat300-1	8	8 (0.00)	< 0.001	163 (266)	8 (0.00)	< 0.001	128 (192)
p_hat300-2	25	25 (0.00)	< 0.001	113 (108)	**25 (0.00)**	< 0.001	**27 (20)**
p_hat300-3	36	36 (0.00)	< 0.001	530 (806)	36 (0.00)	< 0.001	633 (1 232)
p_hat700-1	11	11 (0.00)	0.010 (0.010)	2 014 (3 276)	**11 (0.00)**	**0.010 (0.020)**	**1 336 (1 898)**
p_hat700-2	44	44 (0.00)	< 0.001	281 (322)	**44 (0.00)**	< 0.001	**112 (83)**
p_hat700-3	62	62 (0.00)	< 0.001	585 (501)	**62 (0.00)**	< 0.001	**219 (282)**
p_hat1500-1	12	12 (0.00)	1.330 (1.720)	207 900 (269 230)	**12 (0.00)**	**1.690 (2.090)**	**145 400 (180 920)**
p_hat1500-2	65	**65 (0.00)**	**0.000 (0.010)**	**784 (1 061)**	65 (0.00)	0.010 (0.010)	331 (1 244)
p_hat1500-3	94	**94 (0.00)**	**0.000 (0.010)**	**1 867 (2 610)**	94 (0.00)	0.010 (0.010)	1 253 (1 498)

Table 4.2: Algorithm comparison on a selected sub-set of the DIMACS benchmark instances. The table shows the median solution quality and within brackets the median percentage deviation from the best known, as well as CPU seconds and steps medians with IQR within brackets. The dominating algorithm is highlighted in bold.

beginning and reused throughout the run. Furthermore some algorithmic improvements to the original RLS have been introduced leading to the final RLS–LTM proposal, including algorithmic and implementation changes. RLS–LTM achieves an order of magnitude difference in CPU times for graphs of reasonable sizes, and the difference appears to grow with the problem dimension. This results drastically changes the overall competitiveness of the Reactive Local Search technique.

The software corresponding to the algorithm, benchmark graphs and the heuristically optimal values are available at request for research purposes.

4.4 Notes on the Published Paper

The results presented in this chapter have been published in [11]. The original design of the algorithm described in the paper set the upper bound on the tabu tenure to a fraction of the estimated maximum clique, i.e., $\mathsf{MAX\_T} = 0.5(|\mathsf{Best}| + 1)$. But a bug that went unnoticed before the publication set the threshold wrongly to $\mathsf{MAX\_T} = |\mathsf{Best}| + 0.5$. In the following chapter we will study the behaviour of RLS–LTM when $\mathsf{MAX\_T} = 0.5(|\mathsf{Best}| + 1)$ as originally designed in the published paper because the results are virtually indistinguishable from those presented in this chapter. Figure 4.2 and 4.3 present the performances of the two algorithms on the DIMACS benchmark set, and a detailed comparison is available in Table 4.3.

Moreover, a change in the algorithm dynamics went undocumented: the original publication [11] does not report that the tabu tenure is never reset during restarts.

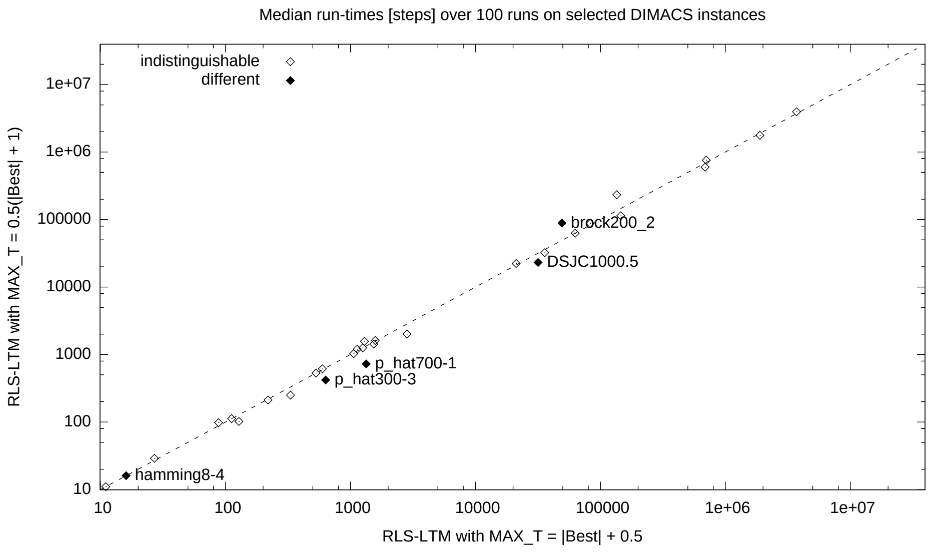


Figure 4.2: Median number of steps to converge to the optimal solution. The empty dots represent the instances for which a Mann-Whitney U-test could not reject the null hypothesis of the algorithms having an identical performance at significance level $\alpha = 0.05$.

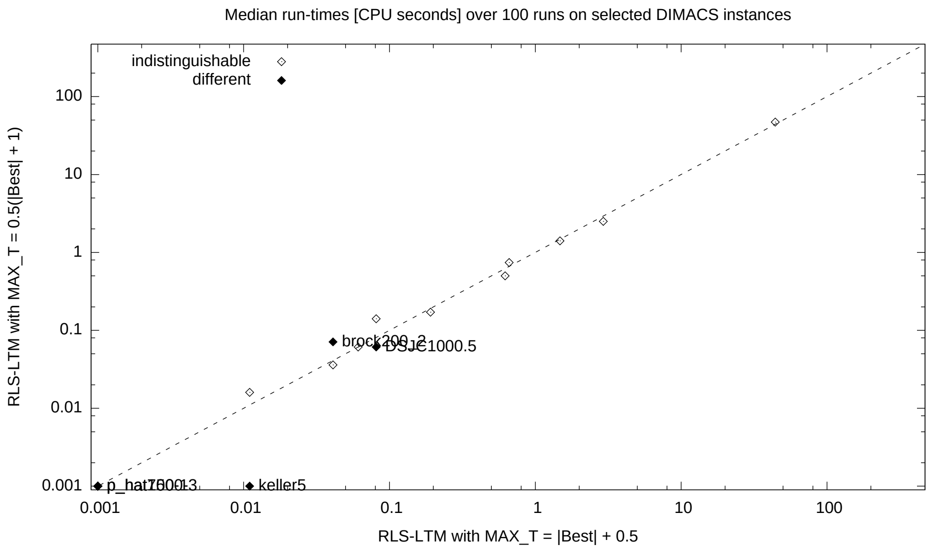


Figure 4.3: Median number of CPU seconds to converge to the optimal solution. The empty dots represent the instances for which a Mann-Whitney U-test could not reject the null hypothesis of the algorithms having an identical performance at significance level $\alpha = 0.05$.

Instance	**Best**	**RLS-LTM with** MAX_T $= 0.5(\lvert\text{Best}\rvert + 1)$			**RLS-LTM with** MAX_T $= \lvert\text{Best}\rvert + 0.5$		
		Solution quality	CPU(s)	Steps	Solution quality	CPU(s)	Steps
C125.9	34	34 (0.00)	0.001 (0.000)	98 (141)	34 (0.00)	0.001 (0.000)	88 (118)
C250.9	44	44 (0.00)	0.001 (0.000)	1 031 (1 124)	44 (0.00)	0.001 (0.000)	1 060 (1 010)
C500.9	57	57 (0.00)	0.061 (0.072)	87 720 (123 830)	57 (0.00)	0.061 (0.130)	82 740 (181 010)
C1000.9	68	68 (0.00)	0.741 (1.115)	756 600 (1 144 600)	68 (0.00)	0.661 (0.937)	703 000 (973 400)
C2000.9	78	78 (0.00)	-	-	78 (0.00)	-	-
DSJC1000.5	15	15 (0.00)	0.061 (0.090)	23 090 (37 950)	15 (0.00)	0.081 (0.133)	31 720 (53 220)
DSJC500.5	13	13 (0.00)	0.001 (0.000)	1 201 (1 454)	13 (0.00)	0.001 (0.010)	1 131 (1 692)
C2000.5	16	16 (0.00)	0.171 (0.253)	32 170 (48 740)	16 (0.00)	0.191 (0.260)	35 760 (49 150)
C4000.5	18	18 (0.00)	47.010 (62.910)	3 956 000 (5 249 000)	18 (0.00)	44.150 (60.790)	3 705 000 (5 113 000)
MANN_a27	126	126 (0.00)	0.036 (0.042)	62 760 (72 060)	126 (0.00)	0.041 (0.040)	62 760 (72 060)
MANN_a45	345	344 (0.29)	-	-	344 (0.29)	-	-
MANN_a81	1099	1 098 (0.09)	-	-	1 098 (0.09)	-	-
brock200_2	12	12 (0.00)	0.071 (0.102)	88 790 (129 660)	12 (0.00)	0.041 (0.060)	49 100 (83 670)
brock200_4	17	17 (0.00)	0.141 (0.265)	233 100 (406 700)	17 (0.00)	0.081 (0.213)	135 000 (346 480)
brock400_2	29	29 (0.00)	-	-	29 (0.00)	-	-
brock400_4	33	33 (0.00)	1.406 (2.660)	1 768 000 (3 223 300)	33 (0.00)	1.476 (1.978)	1 890 000 (2 521 300)
brock800_2	24	21 (14.29)	-	-	21 (14.29)	-	-
brock800_4	26	21 (23.81)	-	-	21 (23.81)	-	-
gen200_p0.9_44	44	44 (0.00)	0.001 (0.000)	1 425 (2 204)	44 (0.00)	0.001 (0.000)	1 535 (1 832)
gen200_p0.9_55	55	55 (0.00)	0.001 (0.000)	613 (668)	55 (0.00)	0.001 (0.000)	596 (562)
gen400_p0.9_55	55	55 (0.00)	0.016 (0.020)	22 270 (37 950)	55 (0.00)	0.011 (0.010)	21 160 (30 150)
gen400_p0.9_65	65	65 (0.00)	0.001 (0.000)	1 574 (1 601)	65 (0.00)	0.001 (0.000)	1 294 (1 132)
gen400_p0.9_75	75	75 (0.00)	0.001 (0.000)	1 614 (1 285)	75 (0.00)	0.001 (0.000)	1 576 (1 184)
hamming8-4	16	16 (0.00)	0.001 (0.000)	16 (0)	16 (0.00)	0.001 (0.000)	16 (0)
hamming10-4	40	40 (0.00)	0.001 (0.000)	529 (1 044)	40 (0.00)	0.001 (0.000)	529 (1 044)
keller4	11	11 (0.00)	0.001 (0.000)	11 (14)	11 (0.00)	0.001 (0.000)	11 (8)
keller5	27	27 (0.00)	0.001 (0.010)	2 007 (4 458)	27 (0.00)	0.011 (0.010)	2 828 (7 150)
keller6	59	59 (0.00)	2.501 (4.556)	594 900 (1 111 900)	59 (0.00)	2.926 (5.464)	686 500 (1 265 300)
p_hat300-1	8	8 (0.00)	0.001 (0.000)	102 (164)	8 (0.00)	0.001 (0.000)	128 (192)
p_hat300-2	25	25 (0.00)	0.001 (0.000)	29 (24)	25 (0.00)	0.001 (0.000)	27 (20)
p_hat300-3	36	36 (0.00)	0.001 (0.000)	418 (695)	36 (0.00)	0.001 (0.000)	633 (1 232)
p_hat700-1	11	11 (0.00)	0.001 (0.010)	725 (1 636)	11 (0.00)	0.001 (0.010)	1 336 (1 898)
p_hat700-2	44	44 (0.00)	0.001 (0.000)	112 (83)	44 (0.00)	0.001 (0.000)	112 (83)
p_hat700-3	62	62 (0.00)	0.001 (0.000)	212 (273)	62 (0.00)	0.001 (0.000)	219 (282)
p_hat1500-1	12	12 (0.00)	0.501 (0.800)	113 600 (189 520)	12 (0.00)	0.621 (0.777)	145 400 (180 920)
p_hat1500-2	65	65 (0.00)	0.001 (0.010)	250 (995)	65 (0.00)	0.001 (0.010)	331 (1 244)
p_hat1500-3	94	94 (0.00)	0.001 (0.010)	1 253 (1 498)	94 (0.00)	0.001 (0.010)	1 253 (1 498)

Table 4.3: Algorithm comparison on a selected sub-set of the DIMACS benchmark instances. The table shows the median solution quality and within brackets the percentage deviation from the best known, as well as CPU seconds and steps medians with IQR within brackets.

Chapter 5

A Comparison of Tabu Search Variations

RLS–LTM has been described in Chapter 4 as a more effective implementation of of the original RLS for MC [13]. Faster restarts and greater diversification allowed to improve over the original RLS for MC, but also caused the explosion of the value of T on some hard instances. In Chapter 4 it is conjectured that with a high value of T it is unlikely that all nodes belonging to the maximum clique are not prohibited and can be added to the current configuration, therefore an upper bound $\mathsf{MAX\_T} = 0.5(|\mathsf{Best}| + 1)$ has been introduced.

In this chapter we present an in-depth analysis of the dynamics of RLS and RLS–LTM. We study how the reactive mechanism impacts on the overall performances, and its effectiveness in tuning the tabu tenure for the instance at hand and the local characteristics of the search landscape.

Appendix A presents a more detailed version of this analysis.

5.1 Peeking Under the Hood of RLS–LTM

Reactive Tabu Search is a meta-heuristic that adapts the tabu tenure T of the underlying Tabu Search. It automatically tunes the parameter T for

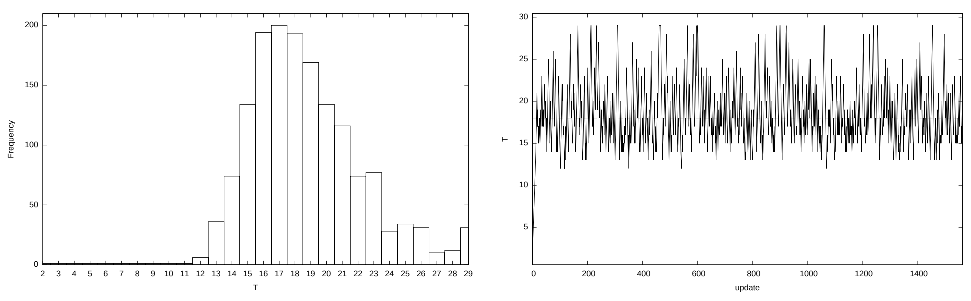


Figure 5.1: Adaptation of tabu tenure T for the instance C500.9. The graph on the left shows how many times the parameter T has been set to a particular value. The graph on the right shows the evolution of the parameter in time. The dashed line shows the median value.

the instance at hand, and more importantly it adapts it throughout the run by reacting on the local characteristics of the search space.

The aim of this study is at understanding how effective is the adaptation of the tabu tenure in the case of RLS–LTM for the MC problem. More specifically we want to understand if RLS–LTM is rapidly converging to the best parameter T for the instance at hand, or if it is adapting it to different values depending on the local characteristic of the search space.

In order to understand how the tabu tenure is adapted, we run the algorithm on the DIMACS benchmark instances and trace the history of the parameter T throughout 1,000,000 iterations. Figure 5.1, 5.2, 5.3, and 5.4 are representative of the four different pictures we got across the benchmark set. In Figure 5.1 the value converges immediately and stays around the average with small oscillations. Also in Figure 5.2 the parameter converges in few iterations but the distribution of T is saturated on the upper-bound MAX_T. The opposite happens in Figure 5.3 where the tenure is updated rarely, and for most of the iterations the parameter T remains on the minimum value allowed. There are very few spikes corresponding to repetitions encountered during the search. On some instances, like in the case of C4000.5 in Figure 5.4, the picture is less clear.

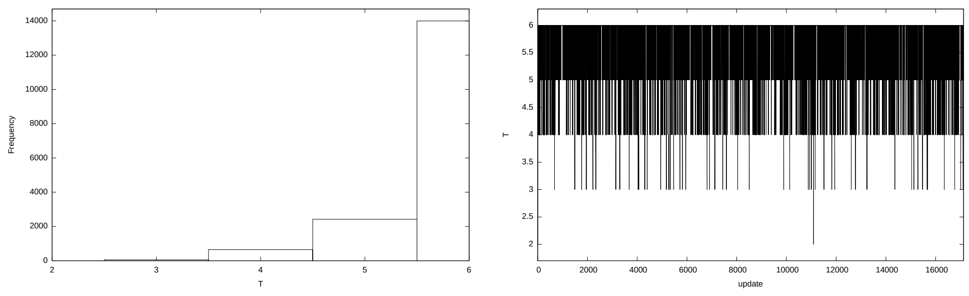


Figure 5.2: Adaptation of tabu tenure T for the instance brock200_4. The graph on the left shows how many times the parameter T has been set to a particular value. The graph on the right shows the evolution of the parameter in time. The dashed line shows the median value.

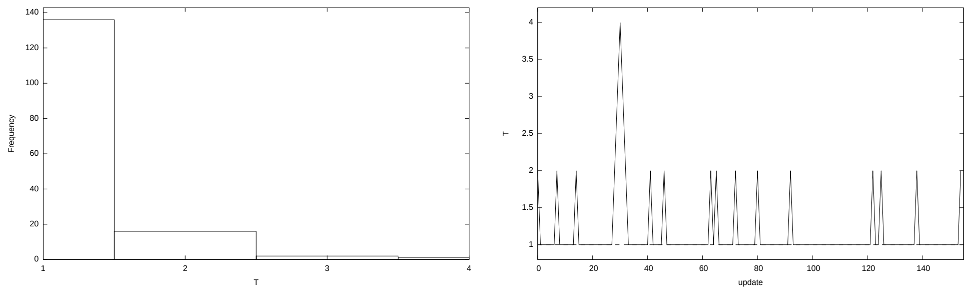


Figure 5.3: Adaptation of tabu tenure T for the instance MANN_a45. The graph on the left shows how many times the parameter T has been set to a particular value. The graph on the right shows the evolution of the parameter in time. The dashed line shows the median value.

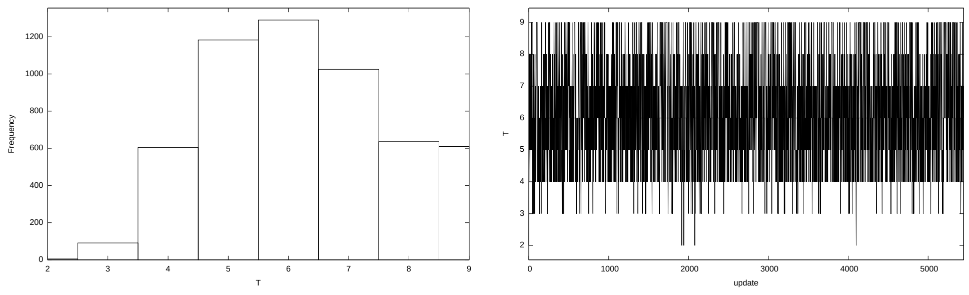


Figure 5.4: Adaptation of tabu tenure T for the instance C4000.5. The graph on the left shows how many times the parameter T has been set to a particular value. The graph on the right shows the evolution of the parameter in time. The dashed line shows the median value.

The overall picture emerging is the following: either there is a strong threshold effect, or the parameter T quickly converges to an average value with little oscillations around it during the search.

Looking at the instances with an evident threshold effect, it is clear that the adaptation of the tabu tenure is not optimal at least in those cases. To better understand how the reactive mechanism impacts on the performances we try to update at each iteration the tenure T to a random value in the interval $[\mathsf{MIN\_T}, \mathsf{MAX\_T}]$. Figure 5.5 shows the performance of a reactive and a random tenure update on the subset of the DIMACS benchmark set where both find the maximum clique on all runs. The maximum computational budget allocated is 100,000,000 iterations. Each dot in the graph represents the median number of iterations over 100 successful runs on a single instance. The performances are highly correlated: the Spearman's rank order test rejects the hypothesis of no significant (monotone) relationship between the samples with p-value $1.67 \cdot 10^{-20}$. The empty dots represent instances for which a Mann-Whitney U-test could not reject the null hypothesis of the algorithms having an identical performance at significance level $\alpha = 0.05$. The white squares on the axes show instances for which the one of the algorithm was not able to find the maximum clique every run. For the other cases the name of the instances is reported. The only notable differences are on the brock200 instances. A Wilcoxon matched pairs signed rank test on the medians could not reject the null hypothesis of the algorithms having an identical performance at significance level $\alpha = 0.05$. The same conclusions can be drawn also looking at the CPU seconds in Figure 5.6. The detailed results are shown in Table 5.1.

The quite surprising results can be explained by the small interval $[\mathsf{MIN\_T}, \mathsf{MAX\_T}]$. Such a small interval makes the reactive mechanism statistically indistinguishable from a Tabu Search selecting a tenure value randomly around $\mathsf{MIN\_T}\frac{\mathsf{MAX\_T} - \mathsf{MIN\_T}}{2}$. The only cases in which the perfor-

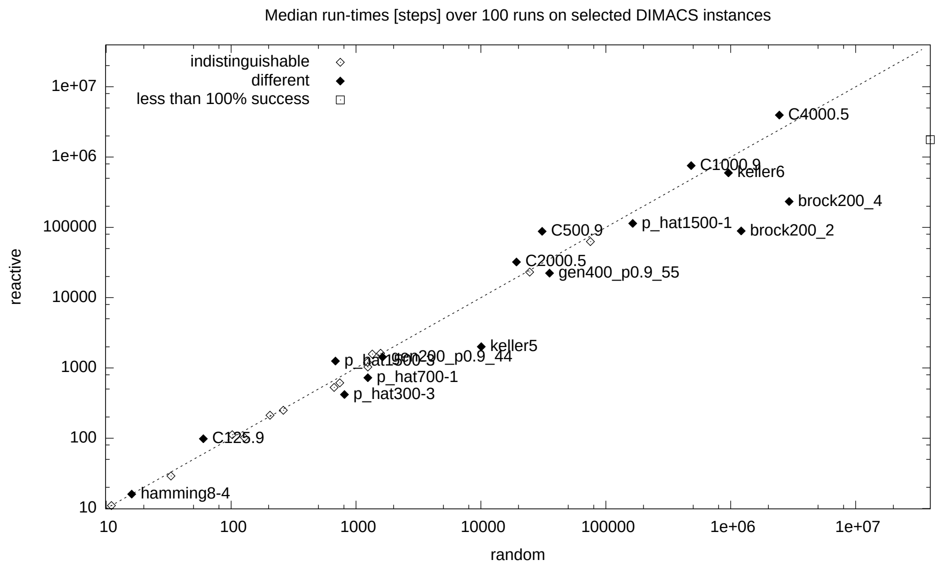


Figure 5.5: Median number of steps to converge to the optimal solution when setting the tenure randomly or reactively. The empty dots represent the instances for which a Mann-Whitney U-test could not reject the null hypothesis of the algorithms having an identical performance at significance level $\alpha = 0.05$. The white squares on the axes show instances for which the one of the algorithm was not able to find the maximum clique every run.

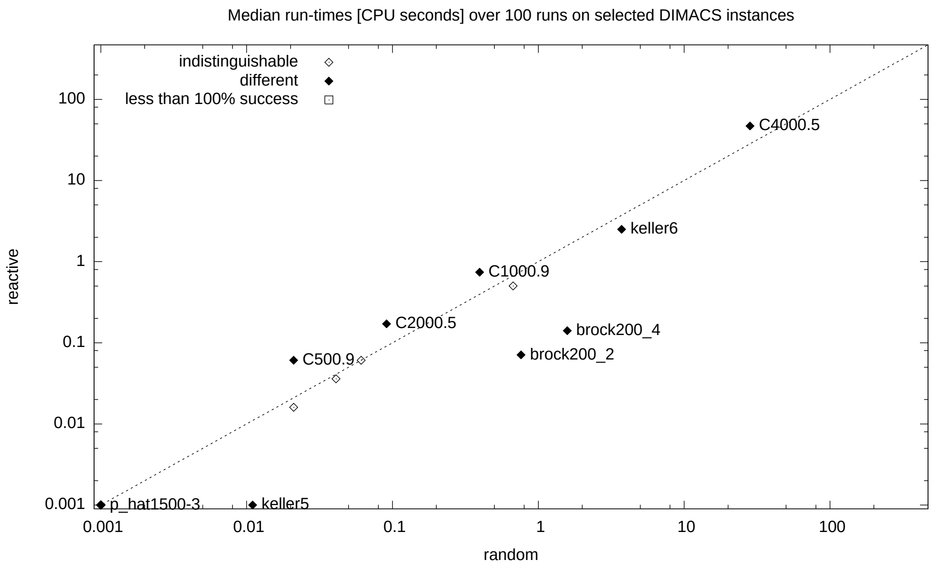


Figure 5.6: Median number of CPU seconds to converge to the optimal solution when setting the tenure randomly or reactively. The empty dots represent the instances for which a Mann-Whitney U-test could not reject the null hypothesis of the algorithms having an identical performance at significance level $\alpha = 0.05$. The white squares on the axes show instances for which the one of the algorithm was not able to find the maximum clique every run.

mance are worse are the brock200 instances in which a stronger diversification would be necessary, and the random distribution of the values of T is centred far from the threshold MAX_T.

Increasing the upper bound MAX_T deteriorates the performances both of RLS–LTM and the algorithm setting the tabu tenure randomly, meaning that both heuristics are strongly sensible to this meta-parameter. This is especially true when the tenure value is adjusted randomly, since the distribution of the tenure values is spread on a larger interval and centred on a value far from the optimum one. The extended version of this chapter in Appendix A has a detailed description of the comparison between the two techniques for different values of MAX_T.

5.2 Long vs. Short Term Memory

The results presented in Chapter 4 show the improved performance of RLS–LTM over RLS both in terms of steps per second and number of steps to converge to the optimum; but it has never been analysed how the algorithmic and the implementation changes contribute to the performance.

We are particularly interested in the algorithmic changes here, since they also introduce undesired effects on the explosion of the tenure T, and the consequent need for the introduction of a new upper bound.

We run RLS–LTM without the algorithmic changes in order to measure the performances of a possible efficient implementation of the original RLS. From Figure 5.8 and 5.9 it emerges that other things being equal the algorithmic changes introduced in RLS–LTM improve the overall performances. The improvement measured on the DIMACS benchmark set and detailed in Table 5.2 is due almost exclusively to the faster restarts. In fact on most instances there is no difference in the median number of steps to reach the optimum, but there is up to an order of magnitude in the

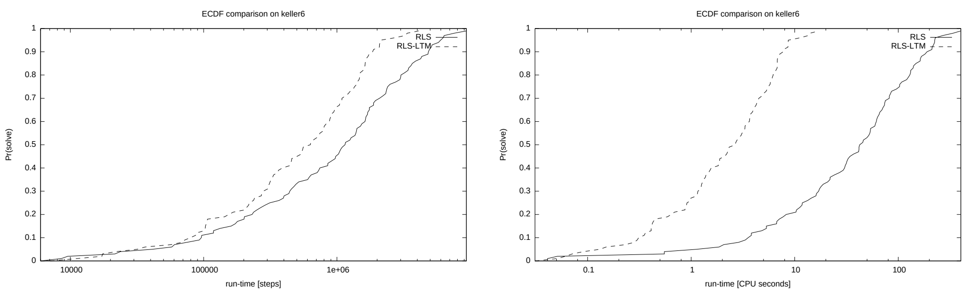


Figure 5.7: Empirical QRTDs on the keller6 instance.

CPU seconds. Figure 5.7 shows a comparison of the the empirical run time distributions on the keller6 instance where RLS–LTM performs in median half of the iterations than RLS in less than 5% of the CPU seconds.

Figure 5.10, 5.11, 5.12, and 5.13 show the adaptation of the tabu tenure of RLS in the four cases discussed at the beginning of this Chapter, namely C500.9, brock200_4, C4000.5, and MANN_a45. Figure 5.12 is almost identical to Figure 5.3 depicting the adaptation of the tenure of RLS–LTM. The small number of repetitions in the search history keeps the tabu tenure on smaller values. Also for the instance C4000.5 (Figure 5.13) RLS keeps the tenure T around small values, but in the case of the hardest brock200_4 (Figure 5.11) the explosion of T before the restarts is quite evident. In the instance C500 depicted in Figure 5.10 the tenure does not explode, and the high number of restarts can be seen from frequency of the smaller values of T.

5.2.1 A Good Tabu Tenure

Figure 5.26(a), 5.26(c), 5.26(f), and 5.26(b) show the mean number of iterations to find the optimum clique over 10 runs for the instances C500.9, brock200_4, MANN_a45, and C4000.5 respectively. The algorithm used to measure the number of iterations is a Tabu search with the same operations

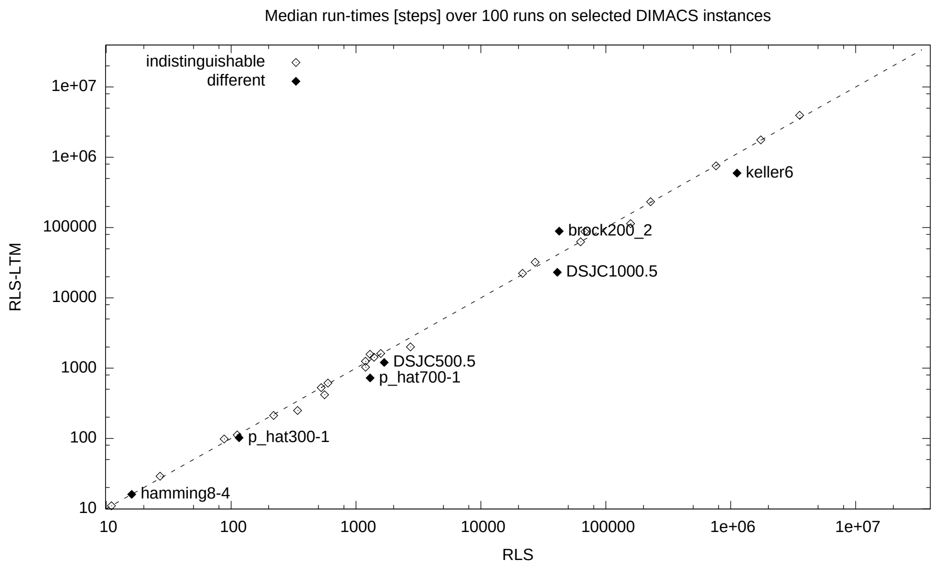


Figure 5.8: Median number of steps to converge to the optimal solution of efficient implementations of RLS and RLS–LTM. The empty dots represent the instances for which a Mann-Whitney U-test could not reject the null hypothesis of the algorithms having an identical performance at significance level $\alpha = 0.05$.

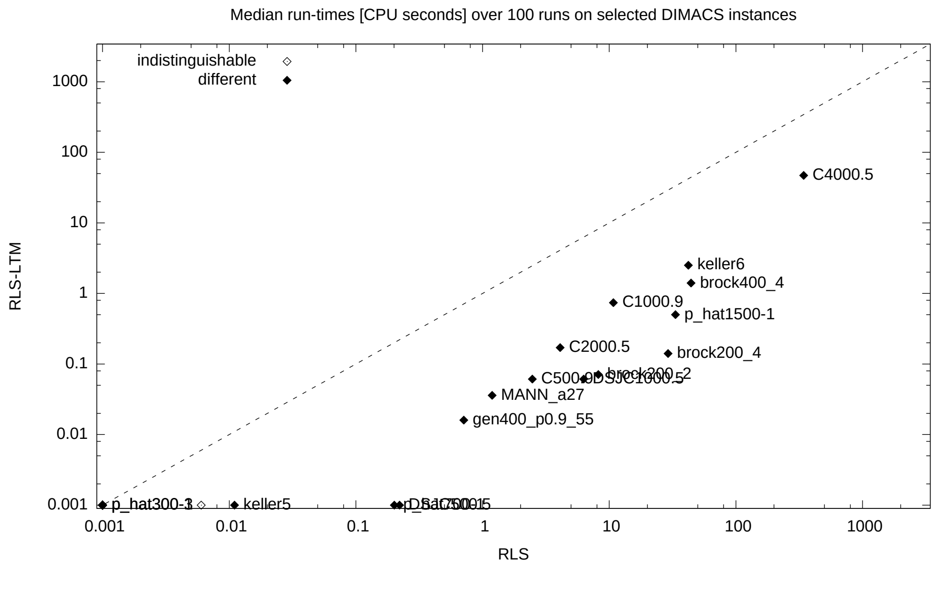


Figure 5.9: Median number of CPU seconds to converge to the optimal solution of efficient implementations of RLS and RLS–LTM. The empty dots represent the instances for which a Mann-Whitney U-test could not reject the null hypothesis of the algorithms having an identical performance at significance level $\alpha = 0.05$.

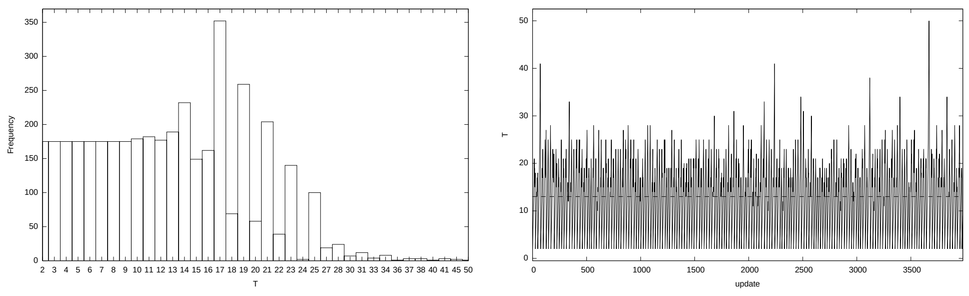


Figure 5.10: Adaptation of tabu tenure T for the instance C500.9. The graph on the left shows how many times the parameter T has been set to a particular value. The graph on the right shows the evolution of the parameter in time. The dashed line shows the median value.

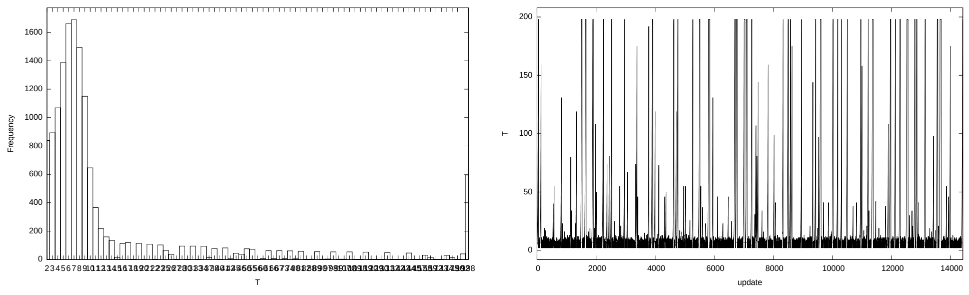


Figure 5.11: Adaptation of tabu tenure T for the instance brock200_4. The graph on the left shows how many times the parameter T has been set to a particular value. The graph on the right shows the evolution of the parameter in time. The dashed line shows the median value.

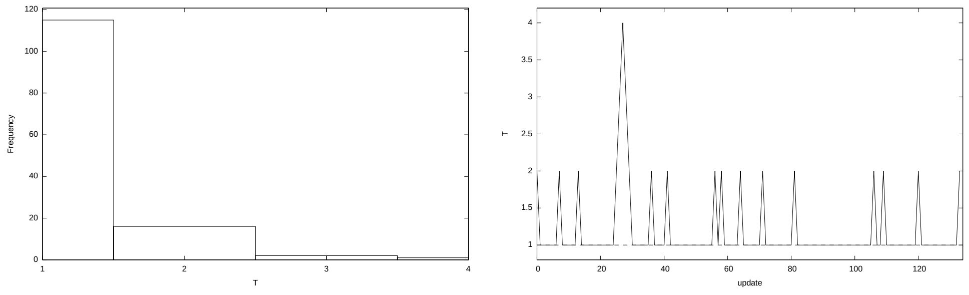


Figure 5.12: Adaptation of tabu tenure T for the instance MANN_a45. The graph on the left shows how many times the parameter T has been set to a particular value. The graph on the right shows the evolution of the parameter in time. The dashed line shows the median value.

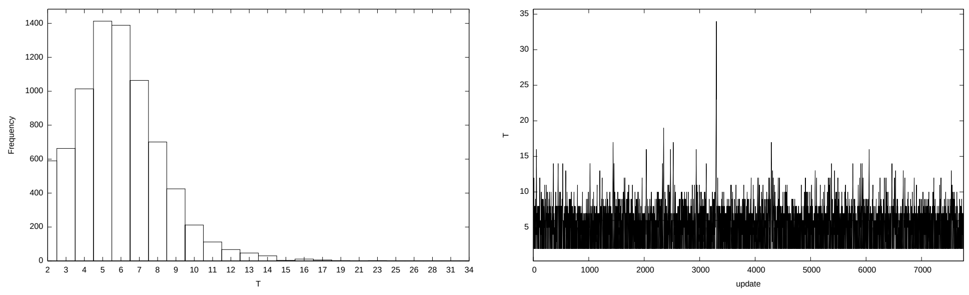


Figure 5.13: Adaptation of tabu tenure T for the instance C4000.5. The graph on the left shows how many times the parameter T has been set to a particular value. The graph on the right shows the evolution of the parameter in time. The dashed line shows the median value.

of RLS (expand, drop, and restart) and a fixed parameter for the tabu tenure T, and for the restart frequency R. The iteration budget in the 10 runs is fixed to 10,000,000. When the algorithm reaches the maximum amount of iterations, it means that it could not find the optimum solution. In most cases there is a setting for the value of T for which the restart frequency is not critical. There are few exceptions, e.g. very hard instances like MANN_a45 in Figure 5.26(f) and MANN_a27 in Figure 5.26(e) in which the opposite is true, i.e., the performances strongly depend on the restart frequency and much less from the value of the tabu tenure T. The restarts make the whole algorithm more robust: except for some extreme values, restarts do not worsen the performances in the instances for which a right value for the tenure is essential, and can be decisive in other instances.

Looking at the histograms in Figure 5.10, 5.11, 5.12, and 5.13 and comparing the mode of the tenure T with the plots in Figure 5.26(a), 5.26(c), 5.26(f), and 5.26(b) it's clear that RLS is able to spot the most appropriate value of T for the particular instance.

The Bias in the Restarts

RLS is able to find a good value of T in most of the cases, but is it because of the effectiveness of the reaction or is it because the frequent restarts mitigate the tenure value?

In [13] restarts were introduced to deal with disconnected part of the search space, but actually without restarts that periodically reset the value of T, RLS could sometimes miss the right value of T for the instance at hand. This is quite noticeable in brock200_4 (Figure 5.11) with the peak on 198 in the histogram (maximum value reachable by T in the experiment $|V| - 2$).

To further investigate this point we run a version of RLS in which there is no upper bound MAX_T and no restarts. Figure 5.15 shows a run on brock200_4 where the explosion of the tenure T is confirmed. The issue is noticeable not only in *small* and hard instances but also on small very *simple* instances where this effect would go unnoticed because of the very few steps required to find the optimum. For example in Figure 5.14 the graph C125.9 has 125 nodes and the maximum clique has size 34, RLS is able to find the max clique in few hundred steps and the explosion of the parameter would be unnoticed. The same can be said for other simple instances, e.g., hamming8-4 in Figure 5.16. Among the same family of instances it happens just on the easiest ones. For hard instances like C4000.5 or keller5 the average value of T determined by RLS (Figure 5.17 and 5.18) lies among the best ones possible for the instance (Figure 5.26(b) and 5.26(d) respectively).

C4000.5 is the most surprising results, RLS–LTM clearly shows the explosion of the tenure and the consequent performance degradation when MAX_T is not small enough (see Section A.1 and Figure A.11 in the appendix). RLS has not this issue as depicted in Figure 5.13. One can

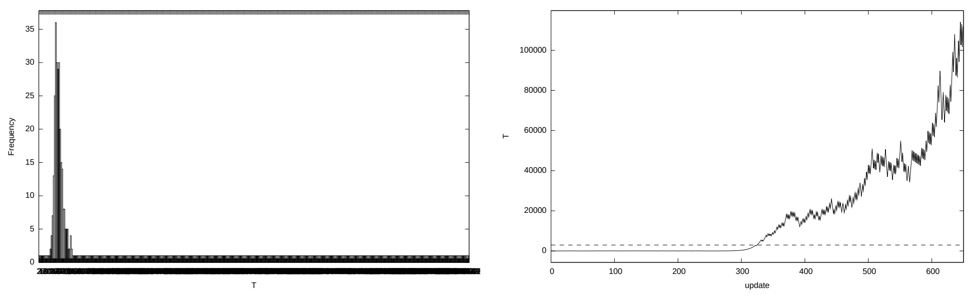


Figure 5.14: Adaptation of tabu tenure T for the instance C125.9 with no restarts and no MAX_T. The graph on the left shows how many times the parameter T has been set to a particular value. The graph on the right shows the evolution of the parameter in time. The dashed line shows the median value.

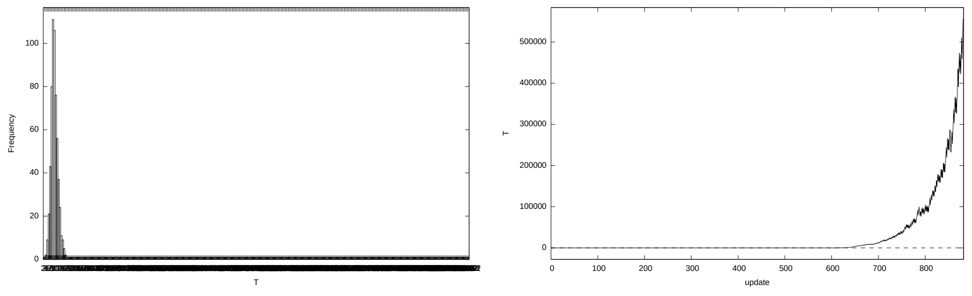
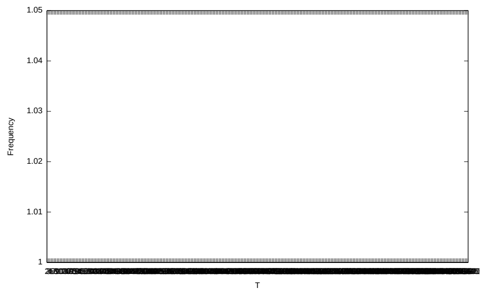


Figure 5.15: Adaptation of tabu tenure T for the instance brock200_4 with no restarts and no MAX_T. The graph on the left shows how many times the parameter T has been set to a particular value. The graph on the right shows the evolution of the parameter in time. The dashed line shows the median value.



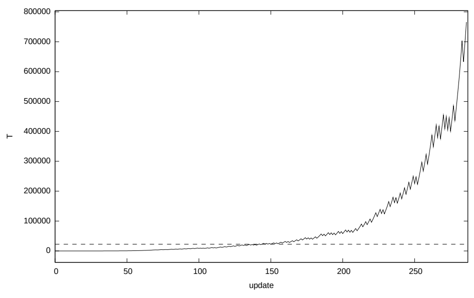
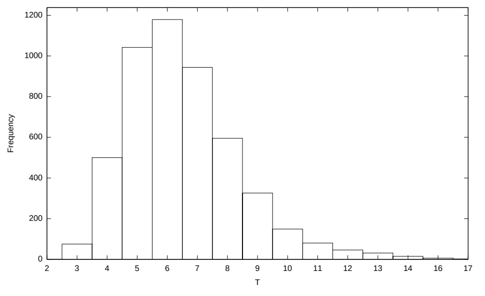


Figure 5.16: Adaptation of tabu tenure T for the instance hamming8-4 with no restarts and no MAX_T. The graph on the left shows how many times the parameter T has been set to a particular value. The graph on the right shows the evolution of the parameter in time. The dashed line shows the median value.



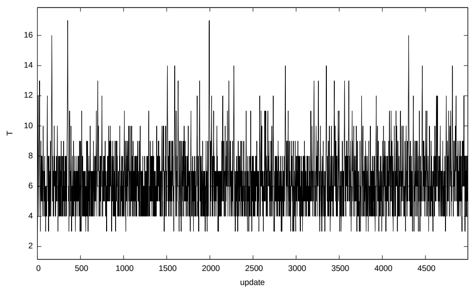


Figure 5.17: Adaptation of tabu tenure T for the instance C4000.5 with no restarts and no MAX_T. The graph on the left shows how many times the parameter T has been set to a particular value. The graph on the right shows the evolution of the parameter in time. The dashed line shows the median value.

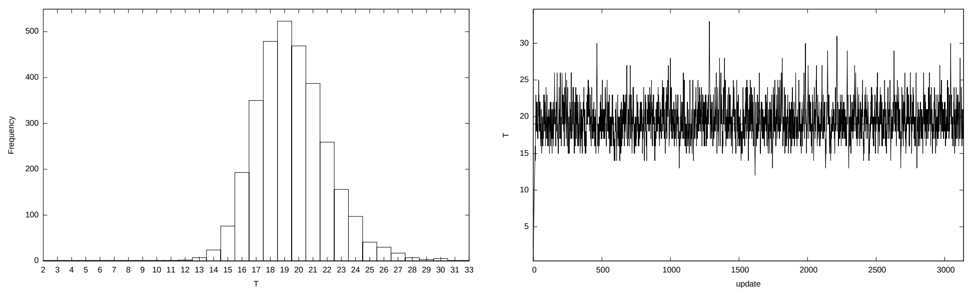


Figure 5.18: Adaptation of tabu tenure T for the instance keller5 with no restarts and no MAX_T. The graph on the left shows how many times the parameter T has been set to a particular value. The graph on the right shows the evolution of the parameter in time. The dashed line shows the median value.

conjecture that the difference is due to the only difference between RLS and RLS–LTM, i.e., the frequent restarts in RLS also reset the tenure T and this prevents the explosion. Oddly, when no upper-bound is set, and without the restarts, the parameter stays centred around good values of T for this instance, see Figure 5.17. This effect does not depend from the upper bound but from the restarts. In fact, adding the restarts leads to the tenure explosion in both cases but in the case of RLS–LTM there is nothing that keeps the value of T to small values like for RLS. The root of this issue lies in the bias in the selection of the node seeding the current configuration. In fact, RLS empties the current configuration during the restarts and seeds it with a node that has never been included in a configuration and with the highest degree. Ties are broken randomly. The problem does not occur if one selects the node seeding the current clique randomly regardless of their degree and regardless of the fact that they have already been part of a solution.

The bias in the restart that is noticeable on certain instances in RLS–LTM could also affect the diversification and therefore the performances of RLS. In order to check if this is the case, we compare the median steps to

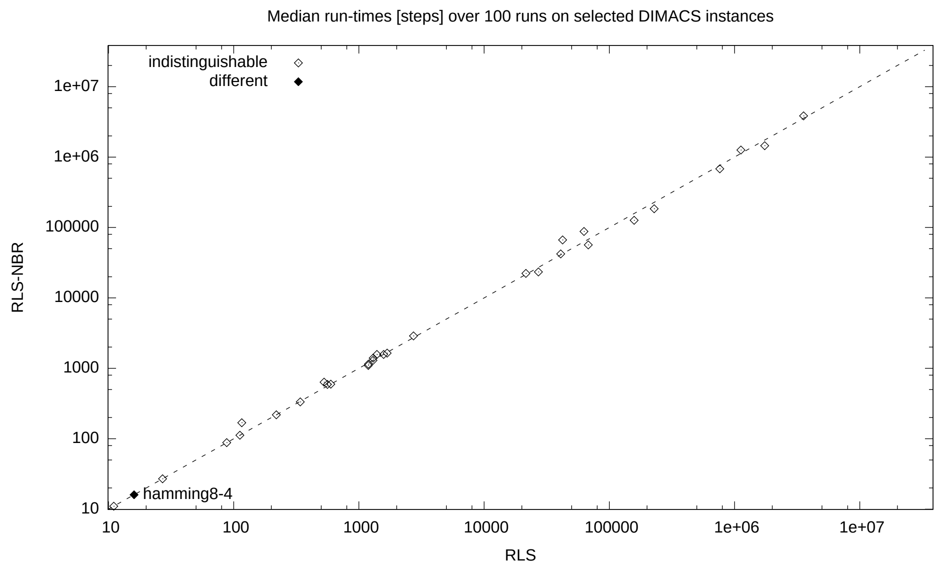


Figure 5.19: Median number of steps to converge to the optimal solution of efficient implementations of RLS and RLS-NBR. The empty dots represent the instances for which a Mann-Whitney U-test could not reject the null hypothesis of the algorithms having an identical performance at significance level $\alpha = 0.05$.

reach the optimum solution for RLS and RLS-NBR. Figure 5.19 compares the two restart implementation showing that the small differences are not statistically significant.

Fixed Tenure T

If we compare RLS with an implementation having the same restart frequency and the best tabu tenure T for the given instance the median steps to converge to the optimal solution is slightly smaller on most instances (see Figure 5.20). The difference in the median CPU seconds in Figure 5.21 is mostly due to the the cost of resetting the hash table during the restarts.

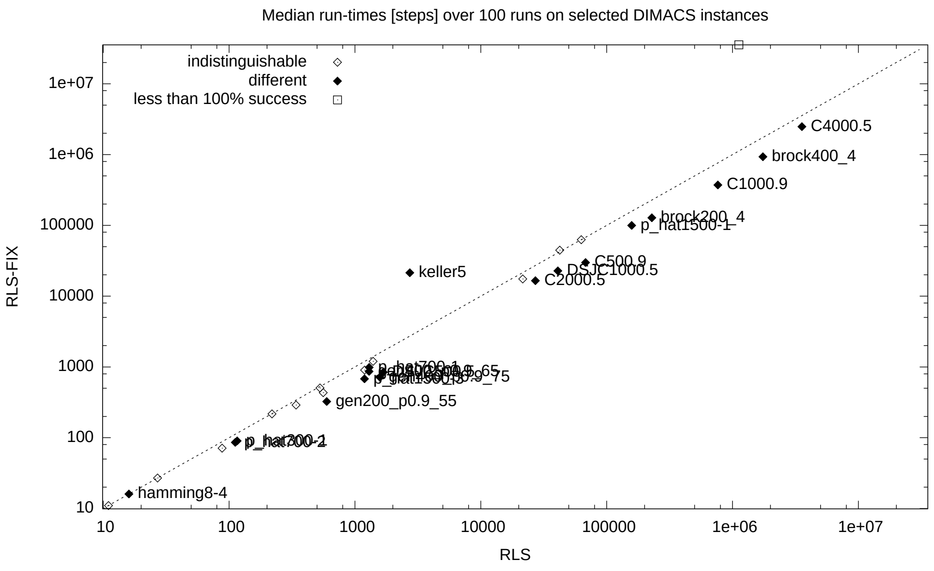


Figure 5.20: Median number of steps to converge to the optimal solution of efficient implementations of RLS and RLS with best fixed T for the instance. The empty dots represent the instances for which a Mann-Whitney U-test could not reject the null hypothesis of the algorithms having an identical performance at significance level $\alpha = 0.05$. The white squares on the axes show instances for which the one of the algorithm was not able to find the maximum clique every run.

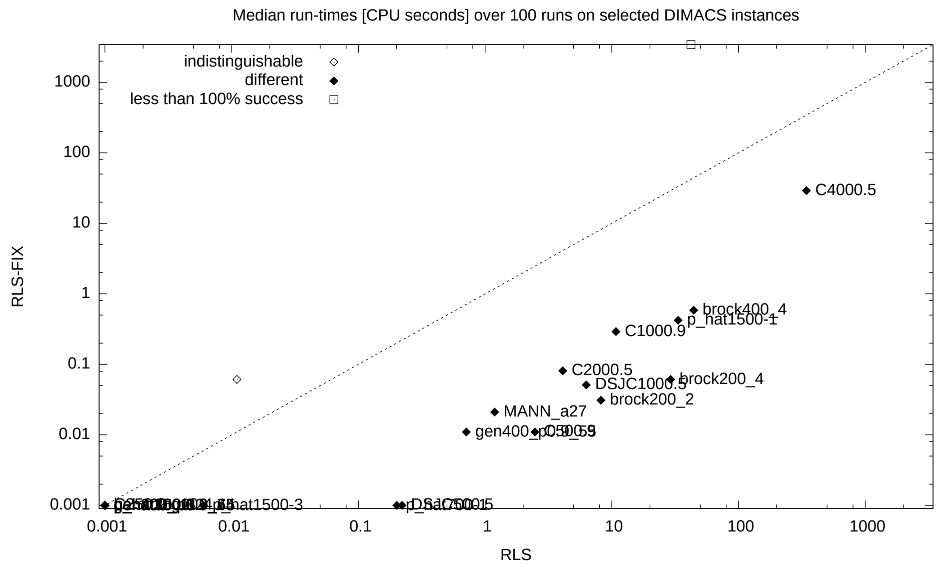


Figure 5.21: Median number of CPU seconds to converge to the optimal solution of efficient implementations of RLS and RLS with best fixed T for the instance. The empty dots represent the instances for which a Mann-Whitney U-test could not reject the null hypothesis of the algorithms having an identical performance at significance level $\alpha = 0.05$. The white squares on the axes show instances for which the one of the algorithm was not able to find the maximum clique every run.

5.3 A New Implementation

We have seen that RLS–LTM can be tricked by too much memory, and without an upper-bound MAX_T the tabu tenure T explodes preventing the algorithm to find the optimum solution. RLS clears the search history and resets the tabu tenure to MIN_T at every restart, therefore it does not show the same parameter explosion as RLS–LTM. Clearing the hash table at every restarts is responsible for worse performances of RLS when compared to RLS–LTM. Even if the median number of steps for the two implementations to find the optimum solution are very similar, the difference in CPU seconds can be of one order of magnitude. Moreover, the bias in the restarts towards highly connected nodes that have never been part of a solution is accountable for the explosion of the tabu tenure in RLS–LTM on some instances. Also in this case, it does not happen in RLS where the restart reset the tabu tenure T.

Building on the analysis presented in the previous sections we build a new implementation RLS–fast in which the restarts are performed seeding the new solution with a completely random node, there is no artificial bound MAX_T, and during the restarts the tabu tenure is reset and the search history cleared in an efficient manner.

The hash table in RLS–LTM starts with 2^{24} elements and if necessary it doubles them every time the fill factor of the table is greater than 0.6. Conflicts are resolved with chaining, and the elements in the chains are are picked up from a pool of pre-allocated memory to avoid expensive system calls for memory allocations during the search.

In RLS–fast the hash table size is fixed to two million elements and does not grow during the search. There is no chaining for resolving conflicts, and locations holding elements older than the last restart are considered empty. Clearing the hash table is as fast as storing the time of the last

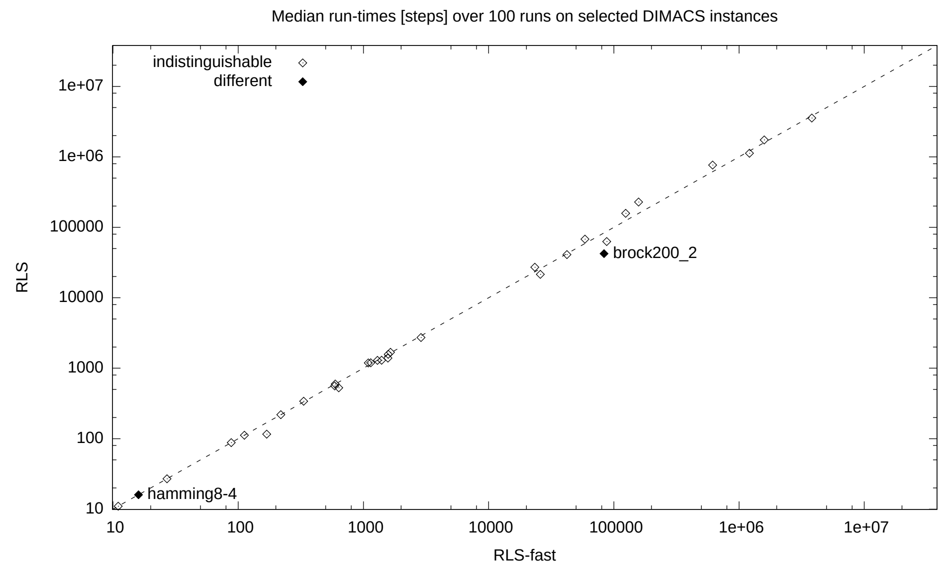


Figure 5.22: Median number of steps to converge to the optimal solution of efficient implementations of RLS and RLS–fast. The empty dots represent the instances for which a Mann-Whitney U-test could not reject the null hypothesis of the algorithms having an identical performance at significance level $\alpha = 0.05$.

restart. Since there is no chaining when the hash table is full a look up operation could be as expensive as $O(n)$ where n is the size of the table. Therefore lookup in the hash table will stop as soon as one expired element is found.

The lookup operations can have false positive since only the hash of the solution are stored in the table, and also false negative because if a solution is stored far from its location because of conflicts there is a probability that after some iterations some location in between will expire.

Figure 5.22 shows how the median number of iteration for reaching the optimum solution for RLS and RLS–fast is for most instances statistically indistinguishable. The faster restarts allow for improving the performances reaching those of RLS–LTM without the undesirable effects of keeping the search history across restarts (see Figure 5.23). Figure 5.24 and 5.25 show the same comparison between RLS–fast and RLS–LTM.

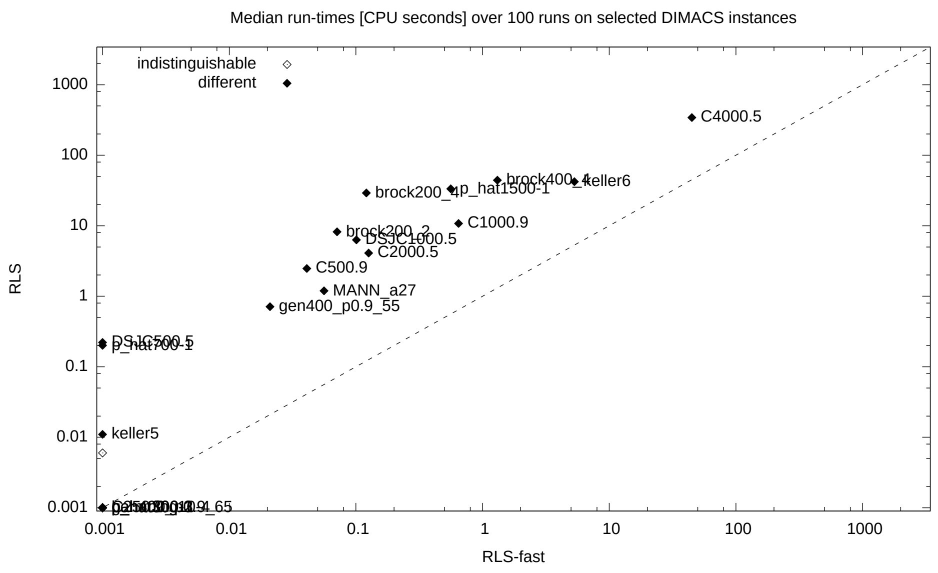


Figure 5.23: Median number of CPU seconds to converge to the optimal solution of efficient implementations of RLS and RLS–fast. The empty dots represent the instances for which a Mann-Whitney U-test could not reject the null hypothesis of the algorithms having an identical performance at significance level $\alpha = 0.05$.

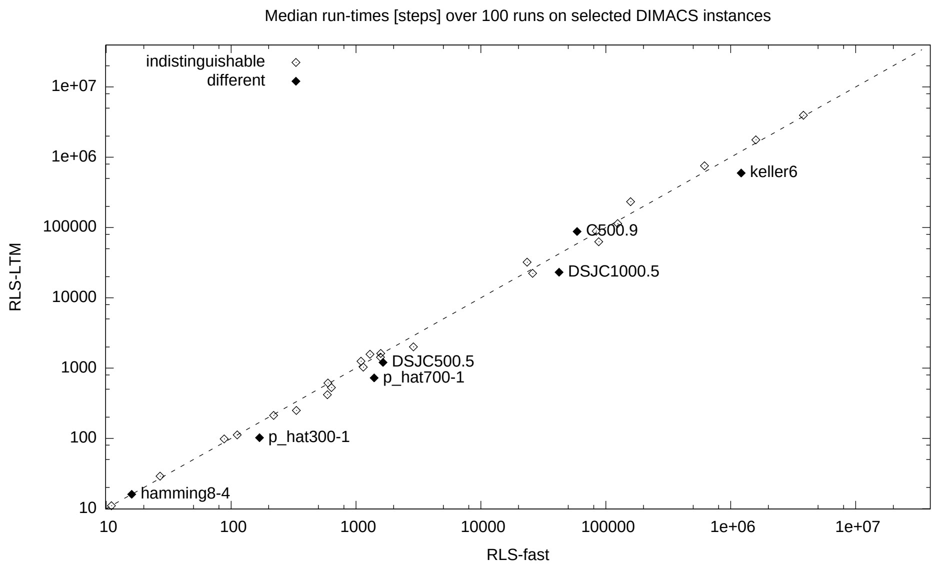


Figure 5.24: Median number of steps to converge to the optimal solution of RLS–LTM and RLS–fast. The empty dots represent the instances for which a Mann-Whitney U-test could not reject the null hypothesis of the algorithms having an identical performance at significance level $\alpha = 0.05$.

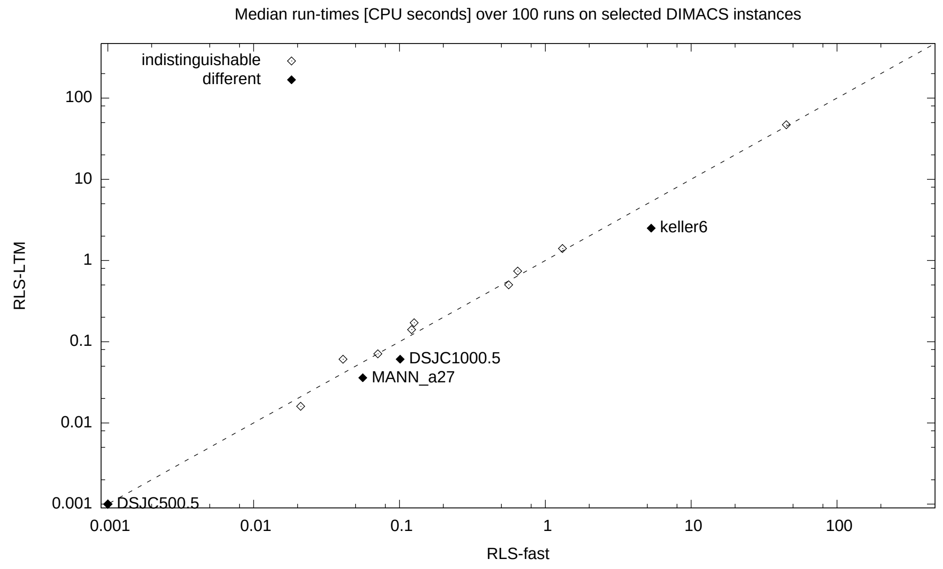


Figure 5.25: Median number of CPU seconds to converge to the optimal solution of RLS–LTM and RLS–fast. The empty dots represent the instances for which a Mann-Whitney U-test could not reject the null hypothesis of the algorithms having an identical performance at significance level $\alpha = 0.05$.

Section A.4 in the appendix, describes a comparison of RLS–fast with Robust Tabu Search [60] (Ro–TS). The results show how the performances are equal and in some cases better than RLS–fast provided that the interval for the tabu tenure is centred around a good value for the instance at hand.

In the experiments the tabu tenure of RoTS is bounded to the size of the current maximum clique, or a fraction of the best value for the instance at hand. We also present the results when the tenure varies around a value which is specific to an instance-family, or when the interval is the same measured in RLS for the instance at hand.

5.4 Conclusions

Looking at the best possible diversification strategy, it seems that on the DIMACS benchmark sets spotting the right value of T leads to the best performances regardless of the restart frequency, with the only notable exception of MANN instances.

A robust tabu search centred around a correct setting for T performs as well as RLS. This result implies that at least on the DIMACS benchmark instances, there is no measurable effect showing that RLS effectively reacts to the local characteristic of the search space.

The speedup in RLS–LTM can be ascribed to the faster restarts that do not need to clear the search history. Avoiding to clear the search history is one of the causes for the explosion of the tabu tenure parameter. Another cause is the bias in the node selection during the restarts. The fact that RLS–LTM also never resets the tabu tenure during the restarts makes the parameter explosion more noticeable.

Knowing the reason for the speedup and for the explosion of the tabu tenure, we implement RLS–fast, which is as efficient as RLS–LTM and does not need for an upper bound for the tabu tenure.

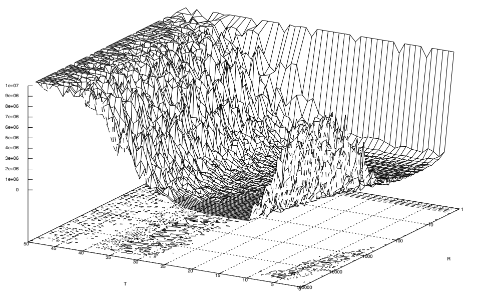
The performances of RLS–fast are comparable to RLS–fix an algorithm that knows a priori the best tabu tenure for every instance in the DIMACS benchmark. This result shows how RLS is able to quickly converge to the best tabu tenure for the instance at hand with very little overhead.

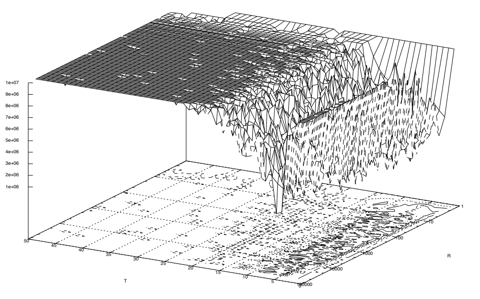
Instance	Best	reactive			random		
		Solution quality	CPU(s)	Steps	Solution quality	CPU(s)	Steps
C125.9	34	34 (0.00)	0.001 (0.000)	98 (141)	34 (0.00)	0.001 (0.000)	60 (50)
C250.9	44	44 (0.00)	0.001 (0.000)	1 031 (1 124)	44 (0.00)	0.001 (0.000)	1 243 (1 842)
C500.9	57	57 (0.00)	0.061 (0.072)	87 720 (123 830)	57 (0.00)	0.021 (0.040)	30 930 (60 320)
C1000.9	68	68 (0.00)	0.741 (1.115)	756 600 (1 144 600)	68 (0.00)	0.396 (0.512)	482 600 (623 000)
C2000.9	78	78 (0.00)	-	-	78 (0.00)	-	-
DSJC1000.5	15	15 (0.00)	0.061 (0.090)	23 090 (37 950)	15 (0.00)	0.061 (0.107)	24 530 (46 164)
DSJC500.5	13	13 (0.00)	0.001 (0.000)	1 201 (1 454)	13 (0.00)	0.001 (0.000)	1 235 (1 906)
C2000.5	16	16 (0.00)	0.171 (0.253)	32 170 (48 740)	16 (0.00)	0.091 (0.110)	19 240 (23 191)
C4000.5	18	18 (0.00)	47.010 (62.910)	3 956 000 (5 249 000)	18 (0.00)	28.280 (38.960)	2 449 000 (3 407 000)
MANN_a27	126	126 (0.00)	0.036 (0.042)	62 760 (72 060)	126 (0.00)	0.041 (0.040)	75 290 (100 050)
MANN_a45	345	344 (0.29)	-	-	344 (0.29)	-	-
MANN_a81	1099	1 098 (0.09)	-	-	1 098 (0.09)	-	-
brock200_2	12	12 (0.00)	0.071 (0.102)	88 790 (129 660)	12 (0.00)	0.761 (1.292)	1 212 000 (2 035 600)
brock200_4	17	17 (0.00)	0.141 (0.265)	233 100 (406 700)	17 (0.00)	1.581 (1.795)	2 936 000 (3 337 000)
brock400_2	29	29 (0.00)	-	-	25 (16.00)	-	-
brock400_4	33	33 (0.00)	1.406 (2.660)	1 768 000 (3 223 300)	33 (0.00)	-	-
brock800_2	24	21 (14.29)	-	-	21 (14.29)	-	-
brock800_4	26	21 (23.81)	-	-	21 (23.81)	-	-
gen200_p0.9_44	44	44 (0.00)	0.001 (0.000)	1 425 (2 204)	44 (0.00)	0.001 (0.000)	1 630 (4 298)
gen200_p0.9_55	55	55 (0.00)	0.001 (0.000)	613 (668)	55 (0.00)	0.001 (0.000)	741 (1 275)
gen400_p0.9_55	55	55 (0.00)	0.016 (0.020)	22 270 (37 950)	55 (0.00)	0.021 (0.013)	35 460 (38 500)
gen400_p0.9_65	65	65 (0.00)	0.001 (0.000)	1 574 (1 601)	65 (0.00)	0.001 (0.000)	1 351 (1 852)
gen400_p0.9_75	75	75 (0.00)	0.001 (0.000)	1 614 (1 285)	75 (0.00)	0.001 (0.000)	1 573 (2 458)
hamming8-4	16	16 (0.00)	0.001 (0.000)	16 (0)	16 (0.00)	0.001 (0.000)	16 (0)
hamming10-4	40	40 (0.00)	0.001 (0.000)	529 (1 044)	40 (0.00)	0.001 (0.000)	668 (1 090)
keller4	11	11 (0.00)	0.001 (0.000)	11 (14)	11 (0.00)	0.001 (0.000)	11 (10)
keller5	27	27 (0.00)	0.001 (0.010)	2 007 (4 458)	27 (0.00)	0.011 (0.020)	10 070 (16 956)
keller6	59	59 (0.00)	2.501 (4.556)	594 900 (1 111 900)	59 (0.00)	3.726 (6.910)	957 500 (1 757 500)
p_hat300-1	8	8 (0.00)	0.001 (0.000)	102 (164)	8 (0.00)	0.001 (0.000)	128 (246)
p_hat300-2	25	25 (0.00)	0.001 (0.000)	29 (24)	25 (0.00)	0.001 (0.000)	33 (20)
p_hat300-3	36	36 (0.00)	0.001 (0.000)	418 (695)	36 (0.00)	0.001 (0.000)	809 (2 136)
p_hat700-1	11	11 (0.00)	0.001 (0.010)	725 (1 636)	11 (0.00)	0.001 (0.010)	1 243 (1 939)
p_hat700-2	44	44 (0.00)	0.001 (0.000)	112 (83)	44 (0.00)	0.001 (0.000)	102 (80)
p_hat700-3	62	62 (0.00)	0.001 (0.000)	212 (273)	62 (0.00)	0.001 (0.000)	205 (238)
p_hat1500-1	12	12 (0.00)	0.501 (0.800)	113 600 (189 520)	12 (0.00)	0.671 (0.971)	164 100 (244 940)
p_hat1500-2	65	65 (0.00)	0.001 (0.010)	250 (995)	65 (0.00)	0.001 (0.010)	262 (709)
p_hat1500-3	94	94 (0.00)	0.001 (0.010)	1 253 (1 498)	94 (0.00)	0.001 (0.010)	685 (1 004)

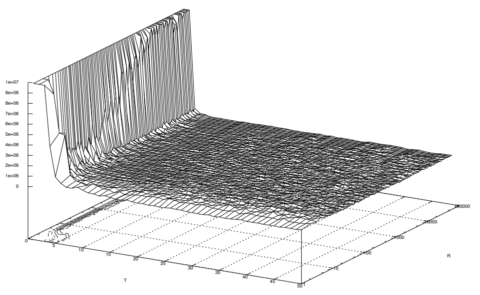
Table 5.1: Algorithm comparison on a selected sub-set of the DIMACS benchmark instances. Threshold MAX_T is set to 0.5(|Best|+1). The table shows the median solution quality and within brackets the percentage deviation from the best known, as well as CPU seconds and steps medians with IQR within brackets.

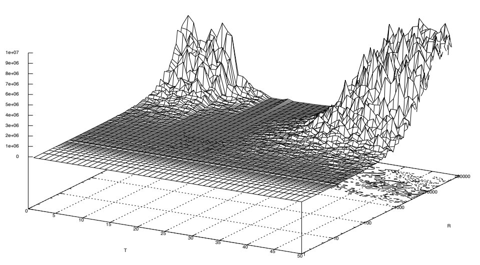
Instance	Best	RLS–LTM			RLS		
		Solution quality	CPU(s)	Steps	Solution quality	CPU(s)	Steps
C125.9	34	34 (0.00)	0.001 (0.000)	98 (141)	34 (0.00)	0.001 (0.000)	88 (150)
C250.9	44	44 (0.00)	0.001 (0.000)	1 031 (1 124)	44 (0.00)	0.001 (0.000)	1 192 (1 081)
C500.9	57	57 (0.00)	0.061 (0.072)	87 720 (123 830)	57 (0.00)	2.471 (3.570)	67 890 (92 280)
C1000.9	68	68 (0.00)	0.741 (1.115)	756 600 (1 144 600)	68 (0.00)	10.780 (15.051)	763 400 (1 052 200)
C2000.9	78	78 (0.00)	-	-	78 (0.00)	-	-
DSJC1000.5	15	15 (0.00)	0.061 (0.090)	23 090 (37 950)	15 (0.00)	6.291 (7.709)	40 930 (48 660)
DSJC500.5	13	13 (0.00)	0.001 (0.000)	1 201 (1 454)	13 (0.00)	0.221 (0.430)	1 681 (1 843)
C2000.5	16	16 (0.00)	0.171 (0.253)	32 170 (48 740)	16 (0.00)	4.096 (6.685)	27 160 (42 750)
C4000.5	18	18 (0.00)	47.010 (62.910)	3 956 000 (5 249 000)	18 (0.00)	342.600 (512.300)	3 556 000 (5 209 000)
MANN_a27	126	126 (0.00)	0.036 (0.042)	62 760 (72 060)	126 (0.00)	1.191 (1.725)	62 760 (90 650)
MANN_a45	345	344 (0.29)	-	-	344 (0.29)	-	-
MANN_a81	1099	1 098 (0.09)	-	-	1 098 (0.09)	-	-
brock200_2	12	12 (0.00)	0.071 (0.102)	88 790 (129 660)	12 (0.00)	8.206 (20.069)	42 370 (106 090)
brock200_4	17	17 (0.00)	0.141 (0.265)	233 100 (406 700)	17 (0.00)	29.120 (52.620)	228 300 (356 500)
brock400_2	29	29 (0.00)	-	-	29 (0.00)	-	-
brock400_4	33	33 (0.00)	1.406 (2.660)	1 768 000 (3 223 300)	33 (0.00)	44.240 (70.090)	1 741 000 (2 749 500)
brock800_2	24	21 (14.29)	-	-	21 (14.29)	-	-
brock800_4	26	21 (23.81)	-	-	21 (23.81)	-	-
gen200_p0.9_44	44	44 (0.00)	0.001 (0.000)	1 425 (2 204)	44 (0.00)	0.001 (0.000)	1 393 (1 982)
gen200_p0.9_55	55	55 (0.00)	0.001 (0.000)	613 (668)	55 (0.00)	0.001 (0.000)	596 (562)
gen400_p0.9_55	55	55 (0.00)	0.016 (0.020)	22 270 (37 950)	55 (0.00)	0.711 (1.245)	21 540 (30 572)
gen400_p0.9_65	65	65 (0.00)	0.001 (0.000)	1 574 (1 601)	65 (0.00)	0.001 (0.000)	1 294 (1 132)
gen400_p0.9_75	75	75 (0.00)	0.001 (0.000)	1 614 (1 285)	75 (0.00)	0.001 (0.000)	1 576 (1 184)
hamming8-4	16	16 (0.00)	0.001 (0.000)	16 (0)	16 (0.00)	0.001 (0.000)	16 (0)
hamming10-4	40	40 (0.00)	0.001 (0.000)	529 (1 044)	40 (0.00)	0.001 (0.010)	527 (848)
keller4	11	11 (0.00)	0.001 (0.000)	11 (14)	11 (0.00)	0.001 (0.000)	11 (4)
keller5	27	27 (0.00)	0.001 (0.010)	2 007 (4 458)	27 (0.00)	0.011 (0.250)	2 727 (4 976)
keller6	59	59 (0.00)	2.501 (4.556)	594 900 (1 111 900)	59 (0.00)	42.130 (86.100)	1 123 000 (2 064 900)
p_hat300-1	8	8 (0.00)	0.001 (0.000)	102 (164)	8 (0.00)	0.001 (0.000)	116 (227)
p_hat300-2	25	25 (0.00)	0.001 (0.000)	29 (24)	25 (0.00)	0.001 (0.000)	27 (20)
p_hat300-3	36	36 (0.00)	0.001 (0.000)	418 (695)	36 (0.00)	0.001 (0.000)	560 (879)
p_hat700-1	11	11 (0.00)	0.001 (0.010)	725 (1 636)	11 (0.00)	0.201 (0.412)	1 298 (1 938)
p_hat700-2	44	44 (0.00)	0.001 (0.000)	112 (83)	44 (0.00)	0.001 (0.000)	112 (83)
p_hat700-3	62	62 (0.00)	0.001 (0.000)	212 (273)	62 (0.00)	0.001 (0.000)	219 (282)
p_hat1500-1	12	12 (0.00)	0.501 (0.800)	113 600 (189 520)	12 (0.00)	33.320 (49.930)	157 800 (226 250)
p_hat1500-2	65	65 (0.00)	0.001 (0.010)	250 (995)	65 (0.00)	0.001 (0.010)	340 (1 288)
p_hat1500-3	94	94 (0.00)	0.001 (0.010)	1 253 (1 498)	94 (0.00)	0.006 (0.010)	1 189 (1 699)

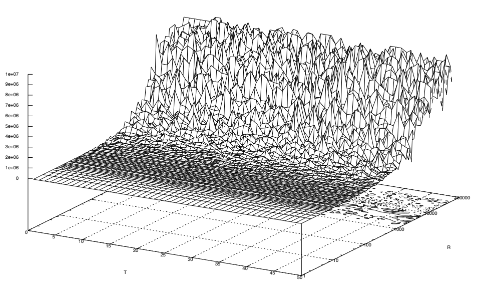
Table 5.2: Algorithm comparison on a selected sub-set of the DIMACS benchmark instances. The table shows the median solution quality and within brackets the percentage deviation from the best known, as well as CPU seconds and steps medians with IQR within brackets.


(a) DIMACS C500.9 instance.


(b) DIMACS C4000.5 instance.


(c) DIMACS brock200_2 instance.


(d) DIMACS keller5 instance.


(e) DIMACS MANN_a27 instance.

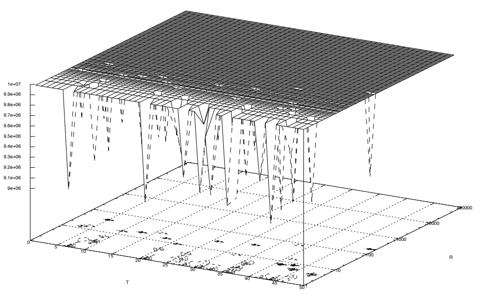

(f) DIMACS MANN_a45 instance.

Figure 5.26: Mean number of iteration to find the optimum solution for different instances. The mean is over 10 runs of 10,000,000 iterations with fixed values for the tabu tenure and restarts frequencies.

Part II

Applications

Chapter 6

Cooperating Local Search

The analysis of the search dynamics and the improved RLS–LTM implementation described in the previous chapters led naturally to the design of a new hyper-heuristic for the MC. Moreover the relatively recent advent of multi-core computers as the standard desktop (and laptop) computer has strengthened the case for the development of parallel hyper-heuristics. This chapter presents the results the research that lead to the design and implementation of Cooperating Local Search (CLS) for the MC problem [55].

6.1 Introduction

From the recent literature, it emerges clearly that there is no single heuristic for the MC problem which dominates on all benchmark instances. Even looking at the two state-of-the art meta-heuristics that have been analysed in the previous chapters, the DIMACS benchmark dataset can be clearly split in families that are solved quickly either by DLS–MC or by RLS–LTM.

As explicitly stated in [54] and shown experimentally in Chapter 2, DLS–MC is quite sensitive to the penalty delay parameter which depends on the instance family and sometimes has to be tuned to sub-family of instances. This issue has been recently rectified by Phased Local Search (PLS) [53]. PLS is a meta-heuristic which goes through three phases during the search: a greedy vertex selection phase, a random selection phase, and a penalty selection phase in which the penalty delay is dynamically adjusted removing the need for extensive parameter tuning. The performances of PLS are comparable to those of DLS–MC.

PLS is clearly the first step towards the design of an algorithm that could perform consistently on all benchmark instances. In fact, by cycling through the Greedy, Random and Penalty phases, it has a larger set of heuristics which could be effective in solving an heterogeneous set of instances.

In this chapter we present Cooperating Local Search (CLS) a novel hyper-heuristic [18] for the MC problem. A Hyper-heuristic is a search algorithm that automates the process of selecting and combining, or generating / adapting low level heuristics for the problem at hand. See [21] for a survey with the most recent developments. Hyper-heuristics can be classified in two main categories: those that select among a set of available low level heuristics and those that generate low level heuristics for the instance at hand. CLS belongs to the first family group.

The relatively recent explosion of availability of multi-core desktop and laptop computers has made the case for the design of a parallel hyper-heuristic for the MC problem. In fact, specialised hardware or clusters of computers are no longer necessary to run a parallel heuristic. On the contrary the design of a parallel algorithm is necessary to fully exploit the computational power of these multi-core architectures.

CLS controls several copies of four low level heuristics that are the most effective for the MC problem. The low level heuristics are run in parallel and are dynamically reallocated depending on the number of available computing cores and more importantly according to the characteristic of the instance at hand. During the search the low level heuristics collect and share information that is exploited to further improve the overall performances. CLS does not depend on exogenous run-time parameters and therefore requires no tuning phase.

6.2 Analysis of the Characteristics of the Benchmark Instances

Looking at Table 4.2 in Chapter 4, it is clear that RLS–LTM and DLS–MC (likewise PLS) can solve effectively all instances of the DIMACS benchmark sets with the notable exceptions of the brock family for RLS–LTM, and keller and MANN families for DLS–MC (and analogously for PLS).

By analysing the instance properties, the two characteristics that better explain the difference in performances are: the size of the plateau areas in the search space, and the vertex degree distribution.

If the size of the plateau area near optimal cliques is high, then an effective heuristic must be able to thoroughly explore the search space without cycling. However this ability could harm the performance on instances with smaller plateaus.

Vertex degree distribution is the second property that can impact on the performances of heuristics using the node degree information differently. A greedy heuristics constructs a clique by selecting the nodes having the highest degree, an example of such greedy heuristic is RLS–LTM. Such strategy is effective when the degree distribution of the vertices belonging to the optimum solution is high relatively to the average degree of the instance. Even without a greedy selection, the vertices that have a higher probability to belong to a clique are those with a high degree. Penalty based algorithm like DLS–MC tend to explore the regions of the search space with vertices having on average a lower degree. This diversification strategy is achieved by penalising the nodes that more frequently are part of maximal solution. The most difficult instances for the Maximum clique appear to be those where the distribution of vertex degrees of the optimum solution corresponds to the distribution of vertex degrees in the instance.

Taking into consideration the two instance characteristics described above, one can divide the DIMACS and the BHOSLIB benchmark instances in the the following categories:

- Most DIMACS instance families have optimum solutions that consists in vertices having degree higher than the instance average degree. Such classes of instances (for example the C family) are tackled effectively by RLS–MC or the greedy phase of PLS.
- A small part of the DIMACS instances (for example the brock family) has maximum cliques whose average degree is smaller than the graph average vertex degree. Those instances are tackled effectively by penalty based algorithms like DLS–MC or the penalty phase of PLS.
- A small part of the DIMACS instances (for example the MANN family) have very large plateau areas near the optimum solutions. Those instances are effectively tackled by algorithms (like RLS-MC) that are able to explore the plateaus while avoiding cycling. In Chapter 5 the analysis of the repeated configurations encountered by RLS is an indirect confirmation of the large plateaus encountered by the algorithm in the MANN family.
- The BHOSLIB instances have optimum solutions with a vertex degree distribution that closely matches that of the whole instance. These instances are hard both for greedy and penalty based heuristics.

In order to be effective on all DIMACS and BHOSLIB instances, a parallel hyper-heuristic should be composed of low level heuristics that focus the search towards set of vertices with different degree distributions. Moreover the heuristics should be able to deal efficiently with the presence or lack of large plateau areas.

6.3 The CLS Hyper-heuristic

CLS is a parallel hyper-heuristic for the MC problem that selects, combines and co-ordinates four low level heuristics

- Greedy Search (GREEDY) which constructs the current solution by selecting randomly among the candidate nodes having highest degree. This heuristic performs limited plateau search.
- Level Search (LEVEL) which selects the candidates vertices among those having highest degree and performs extensive plateau search.
- Penalty Search (PENALTY), which selects the candidate vertices among those having minimum penalty. This low level heuristics focusses on lower degree nodes.
- Focus Search (FOCUS) which focuses on a vertex degree for the current iteration and selects vertices trying to build a clique with an average degree as close as possible to the degree currently on focus. This allows to focus and change the bias dynamically during the search.

The CLS hyper-heuristics goes through two phases. In the first one, at the beginning of the search, the low level heuristics are assigned to computing cores following a predefined schedule. After a short period of time, CLS uses the feedback from the GREEDY heuristic to reconfigure the allocation of the low level heuristics. This reallocation tries to maximise the performances by increasing the number of copies of the LEVEL heuristics in case GREEDY heuristic detected large plateau areas.

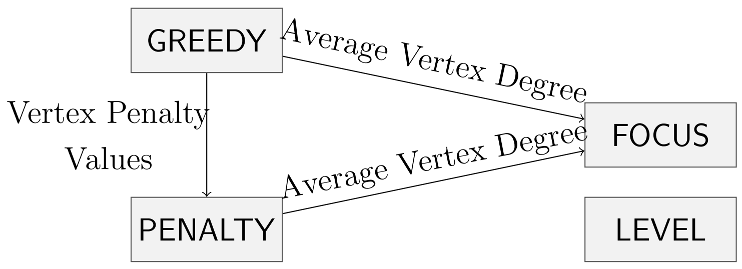


Figure 6.1: Sharing of information between the CLS heuristics.

Figure 6.1 depicts how the information is shared among the low level heuristics. PENALTY heuristic uses the penalties accumulated by GREEDY. Moreover, by measuring the average vertex degree of the configurations visited by GREEDY and PENALTY, the FOCUS heuristic is directed toward areas of the search space not already covered by other low level heuristics.

CLS pseudocode is depicted in Listing 6.1. It alternates between an iterative improvement phase (lines 7–11) and a plateau search phase (lines 12–14). In the former a maximal solution is constructed by repeatedly expanding the current configuration C by selecting a candidate vertex from PossibleAdd. In the latter, a vertex from OneMissing is swapped with a non adjacent vertex in C. When PossibleAdd $= \emptyset$ and OneMissing $= \emptyset$ the search terminates for GREEDY, FOCUS and PENALTY. LEVEL continues with an extensive plateau search using a reactive tabu heuristic to prevent cycling. At this point C is perturbed and the search restarted from a different initial configuration.

The different low level heuristic define their behaviour in the Select, Continue, and Restart functions:

- Select:
  - GREEDY and LEVEL select uniformly random from the vertices having highest degree in the subgraph induced by PossibleAdd. When selecting from the OneMissing set, GREEDY selects a ver-

Listing 6.1: CLS hyper-heuristic pseudo-code.

```
Input: integer $tcs$ (target clique size); $max\text{-}time$
Output: Clique of cardinality $tcs$ or 'failed'
function CLS ($tcs$, $max\text{-}time$)
    $C \leftarrow$ random $v \in V$
    do
        do
            while PossibleAdd $\neq \emptyset$ do
                $v \leftarrow$ Select(PossibleAdd)
                $C \leftarrow C \cup \{v\}$
                if $|C| = tcs$
                    return $C$
            if OneMissing $\neq \emptyset$
                $v \leftarrow$ Select (OneMissing)
                $C \leftarrow [C \cup \{v\}] \setminus \{i\}$, where $\{i\} = C \setminus N(v)$
        while Continue
    Restart
    while $time < max\text{-}time$
    return 'failed'
```

tex uniformly random among those having highest degree, while LEVEL selects the vertex which causes the maximum increase in |PossibleAdd| (like in RLS–MC).

  - FOCUS selects uniformly random among the candidate vertices in whose degree is the closest to vertex degree which is on focus.
  - PENALTY selects uniformly random among the candidate vertices having minimum vertex penalty.

- Continue: GREEDY, FOCUS and PENALTY will continue to explore the search landscape until PossibleAdd is empty, and OneMissing is empty or contains vertices that have already been in C (like in DLS–MC). LEVEL keeps exploring the search space by extensive plateau search using a reactive tabu mechanism to avoid cycles (like RLS–MC).

- Restart: Perturbation is performed by adding a random vertex $v \in V$ to C and removing from C all vertices not connected to v. This allows to have bigger cliques immediately after the restart.

The four low level heuristics coordinated by CLS have been devised to be complementary and share useful information during the search. The design rationale and the information shared by each heuristic is the following:

- GREEDY: This heuristic is particularly effective on the instances where the search space is rugged, i.e., it has a small amount of plateau areas, and the maximum clique has a high average vertex degree. Moreover it collects penalties that are passed to the PENALTY heuristic forcing it to explore a part of the search space not already visited by GREEDY.

- LEVEL: This heuristic uses a greedy selection method like GREEDY but it also performs an extensive plateau search using a reactive tabu technique to detect and avoid cycles.

- PENALTY: This heuristic uses a selection method that exploits the penalties created by GREEDY. This diversification is useful when the maximum clique has few or even no vertices in common with the current solution C. In these cases without penalties even with a strong restart there is the risk that the heuristic guidance function of GREEDY leads the hyper heuristic towards the same solutions visited before the restart. In Section 5.2.1 we have analysed one of the problems that could arise when restart are not able to diversify properly.

- FOCUS: This heuristic targets areas of the search space that the three other heuristics tend to ignore. The average vertex degrees of the configurations visited by GREEDY and PENALTY is fed to FOCUS so that it focuses on constructing cliques having an average vertex degree which is different from the ones produced by GREEDY and PENALTY. The focus average degree is updated at each Restart.

The initial allocation of heuristics to cpu cores is shown in Table 6.1. All heuristics run in parallel until a specified solution quality is matched or the specified maximum amount of selections is reached. After after $1,000,000$ selections GREEDY determines if there are large plateau areas and forces the hyper-heuristic to reallocate the low level heuristics to a Plateau or Non-Plateau configuration. These configurations remain fixed until the end of the search.

We have developed two versions of CLS, both implemented in C / C++. The first requires LAM / MPI ([19], [59]) software to run it on either a single machine or a cluster of machines. In the latter case, all computing cores of each machine in the cluster can be exploited by CLS. All low level heuristics are independent and communicate asynchronously by means of the MPI library. The second implementation is based on pthreads on Unix and can

Number of available cores	Core Number	CLS Configuration		
		Initial	Plateau	Non-Plateau
1	1	GREEDY, PENALTY LEVEL, FOCUS	GREEDY, PENALTY LEVEL, FOCUS	GREEDY, PENALTY LEVEL, FOCUS
2	1	GREEDY, PENALTY	GREEDY, PENALTY	GREEDY, PENALTY
	2	LEVEL, FOCUS	LEVEL, FOCUS	LEVEL, FOCUS
3	1	GREEDY, PENALTY	GREEDY, PENALTY	GREEDY, PENALTY
	2	LEVEL	LEVEL, FOCUS	LEVEL, FOCUS
	3	FOCUS	LEVEL	FOCUS
4	1	GREEDY	GREEDY, PENALTY	GREEDY, PENALTY
	2	PENALTY	LEVEL, FOCUS	LEVEL, FOCUS
	3	LEVEL	LEVEL	FOCUS
	4	FOCUS	LEVEL	FOCUS
> 4	1	GREEDY	GREEDY, PENALTY	GREEDY, PENALTY
	2	PENALTY	LEVEL, FOCUS	LEVEL, FOCUS
	3	LEVEL	LEVEL	FOCUS
	4	FOCUS	LEVEL	FOCUS
	> 4	FOCUS	LEVEL	FOCUS

Table 6.1: Mappings of CLS heuristics to cores. Heuristics sharing a core run with the same priority.

be run on a single multicore machine. Also in this case all heuristics run independently and communicate by means of a synchronised shared data structure. All results presented in the next sections are from the LAM / MPI version. Both implementations have comparable performances.

6.4 Empirical Performance Results

We evaluate the performances of CLS on the *Second DIMACS Implementation Challenge (1992–1993)* [1], which has been used throughout the literature to assess the performances of MC heuristics.

We also assess the performances by means of the *Benchmarks with Hidden Optimum Solutions for Graph problems*[2] (BHOSLIB). The MC instances in this benchmark are transformed from satisfiable SAT instances of Model RB. This allows to hide an exact solution in a large random graph, with the hidden maximum clique having a vertex degree distribution which contains many low and high degree vertices (see Figure 6.2).

The empirical assessment of the performances has been performed on

[1]http://dimacs.rutgers.edu/Challenges/

[2]http://www.nlsde.buaa.edu.cn/~kexu/benchmarks/graph-benchmarks.htm

a machine that executing the DIMACS Maximum Clique Machine Benchmark[3] requires 0.24 CPU seconds for r300.5, 1.49 CPU seconds for r400.5 and 5.65 CPU seconds for r500.5. On this reference machine four out of the eight computing cores have been allocated to CLS.

6.4.1 CLS Performance

To evaluate the performance on both benchmark sets, we run the CLS hyper-heuristic for 100 independent trials on each instance. The trials terminate after a fixed number of iterations or when the optimum solution for the instance is reached. In case no such value is known, the best putative maximum clique in literature is used as target clique size. There are three exception for this second case namely C2000.9, MANN_a45 and MANN_a81. In these cases we let CLS run for a longer amount of iterations for 10 trials in order to give an estimate of the hardness of these instances for CLS.

Table 6.2 shows the results of CLS on the DIMACS benchmark for those instances in which it required more than 0.01 seconds to find the optimum with a 100% success rate. CLS is able to solve all 80 benchmark instances with a 100% success rate for all the 80 DIMACS instances with an average runtime of less than 1 second for 72 of the 80 instances. For 3 of the remaining 8 instances, having at least 800 vertices, the runtime is greater than 10 seconds.

On the C2000.9 instance CLS found maximal cliques of size 78 for all 100 trials. The best solution known in literature has size 80 [33]. On the MANN_a45 instance CLS finds cliques of size 344 on 100 trials while [37] and [13] are able to find cliques of size 345. The same can be said with and MANN_a81 where all 100 trials achieved 1098 as compared to 1099 [41].

CLS is also able to find the maximal clique with 79 vertices in C2000.9

[3] *dmclique*, ftp://dimacs.rutgers.edu in directory /pub/dsj/clique

Instance	ω	**PLS**		**CLS**	
		Success	CPU(s)	Success	RT
brock200_2	12	100	0.01	100	0.01
brock200_4	17	100	0.03	100	0.05
brock400_1	27	100	0.42	100	0.44
brock400_2	29	100	0.15	100	0.11
brock400_3	31	100	0.07	100	0.05
brock400_4	33	100	0.04	100	0.03
brock800_1	23	100	11.73	100	2.63
brock800_2	24	100	9.51	100	2.25
brock800_3	25	100	5.88	100	1.64
brock800_4	26	100	2.55	100	0.76
c1000.9	68	100	0.73	100	0.18
c2000.5	16	100	0.28	100	0.09
c2000.9	78	100	43.96	100	9.47
c4000.5	18	100	58.31	100	17.73

Instance	ω	**PLS / RLS**		**CLS**	
		Success	CPU(s)	Success	RT
c500.9	57	100	0.07	100	0.04
dsjc1000.5	15	100	0.18	100	0.05
gen400_p0.9_55	55	100	0.10	100	0.04
keller6*	59	100	7.16	100	1.19
MANN_a27	126	100	0.01	100	0.03
MANN_a45*	344	100	35.24	100	20.03
MANN_a81*	1098	100	64.64	100	27.04
p_hat1500-1	12	100	1.28	100	0.48
san1000	15	100	1.84	100	0.53
san400_0.5_1	13	100	0.02	100	0.01
san400_0.7_1	40	100	0.02	100	0.01
san400_0.7_2	30	100	0.04	100	0.01
san400_0.7_3	22	100	0.07	100	0.06

Table 6.2: CLS performance results, averaged over 100 successful independent trials, for the DIMACS benchmark instances where the average run time was greater than 0.01 seconds. For all the remaining 53 DIMACS benchmark instances, CLS achieved a 100% success rate in less than 0.01 seconds. The 'CPU(s)' column is the PLS CPU time in seconds from [53] scaled to the reference machine while 'RT' is the CLS run-time in seconds. The CPU(s) column for the instances followed by '*' correspond to RLS–LTM run on the reference machine.

in 10 out of 10 trials with an average run time of 273.71 seconds. It finds the clique of 80 vertex in C2000.9 in 1 of 10 trials with a maximum run time allowed for each trial of 2150 seconds. The 345 vertex clique is found for MANN_a45 in 1 out of 10 trials with a maximum runtime of 2000 seconds per trial. For the MANN_a81 instance, CLS is able to find the clique with 1099 nodes in 1 out of 10 trials with a maximum runtime of 5800 seconds for each trial.

Table 6.3 shows the results averaged over 100 trials on the BHOSLIB benchmark instances. The only other algorithm in literature that has been tested on the same benchmark is PLS [53]. The performances of PLS are reported in Table 6.3.

Looking at the most difficult instances for which both CLS and PLS are able to find the optimum solution on all 100 trials, CLS outperforms PLS even if we divide the CPU seconds of PLS by four, which is the number of cores utilised by PLS. More importantly CLS is able to solve more instances with a 100% success rate. For the frb100-40 instance, CLS is able

Instance	ω	PLS		CLS		
		Success	CPU(s)	Success	RT	FT
30-15-1	30	100	0.05	100	0.01	
30-15-2	30	100	0.07	100	0.01	
30-15-3	30	100	0.53	100	0.17	
30-15-4	30	100	0.03	100	0.00	
30-15-5	30	100	0.25	100	0.12	
35-17-1	35	100	3.20	100	1.81	
35-17-2	35	100	0.95	100	0.30	
35-17-3	35	100	0.20	100	0.04	
35-17-4	35	100	4.34	100	1.01	
35-17-5	35	100	0.61	100	0.19	
40-19-1	40	100	2.55	100	0.55	
40-19-2	40	100	41.59	100	11.20	
40-19-3	40	100	3.71	100	0.83	
40-19-4	40	100	17.72	100	8.69	
40-19-5	40	100	76.67	100	12.49	
45-21-1	45	100	31.79	100	7.28	
45-21-2	45	100	63.50	100	12.74	
45-21-3	45	100	318.27	100	17.04	
45-21-4	45	100	45.57	100	8.42	
45-21-5	45	100	83.70	100	10.98	
50-23-1	50	72	803.75	100	114.51	
50-23-2	50	45	900.27	99	270.07	1510
50-23-3	50	16	800.53	36	589.38	1420
50-23-4	50	100	97.20	100	11.49	
50-23-5	50	99	335.38	100	36.19	
53-24-1	53	6	1312.48	19	683.61	1550
53-24-2	53	23	1190.64	100	291.86	
53-24-3	53	66	911.44	100	211.66	
53-24-4	53	46	1094.08	90	469.25	1650
53-24-5	53	85	753.22	100	55.19	
56-25-1	56	12	953.34	94	613.35	1800
56-25-2	56	6	1308.64	87	562.36	1700
56-25-3	56	8	1135.08	97	478.29	1800
56-25-4	56	68	1002.42	100	97.60	
56-25-5	56	81	837.18	100	246.55	
59-26-1	59	0	$--$	29	751.56	1600
59-26-2	59	0	$--$	6	774.57	1900
59-26-3	59	6	1482.88	46	804.35	1850
59-26-4	59	5	1571.94	33	780.22	1900
59-26-5	59	78	917.24	100	153.65	

Table 6.3: CLS performance results, averaged over 100 successful independent trials, for the BHOSLIB benchmark instances. The 'CPU(s)' column is the PLS CPU time in seconds from [53] scaled to the reference machine while 'RT' is the CLS run-time in seconds. The 'FT' column denotes the maximum number of seconds allocated to CLS.

Instance	SAT 2004	CLS
	Results	Success
frb40-19-1	Solved by 28 solvers	100
frb40-19-2	Solved by 27 solvers	100
frb45-21-1	Solved by 8 solvers	100
frb45-21-2	Solved by 5 solvers	100
frb50-23-1	Solved by 1 solver	100
frb50-23-2	Solved by 1 solver	99
frb53-24-1	Unsolved	19
frb53-24-2	Unsolved	100
frb56-25-1	Unsolved	94
frb56-25-2	Unsolved	87
frb59-26-1	Unsolved	29
frb59-26-2	Unsolved	6

Table 6.4: CLS performance compared to 55 SAT solvers of the SAT Competition 2004.

to find a clique of size 97 (optimum has size 100). Some corresponding SAT instances were used in the SAT Competition 2004. The results of CLS are an improvement also over the 55 SAT solvers partecipating whose results are reported in Table 6.4.

These results demonstrate that CLS achieves excellent and robust performance on both the DIMACS and the BHOSLIB benchmark datasets. See [54, 53] for comparisons among other state-of-the-art heuristics.

Since we run CLS on four cores in all the test of our empirical assessment, we present in Table 6.5 a comparison with all possible combinations of RLS and PLS running in parallel with different random initiali-

Instance	RRRR	RRRP	RRPP	RPPP	PPPP	CLS	GREEDY	LEVEL	FOCUS	PENALTY
C2000.9	9.55	11.40	12.12	23.49	49.30	9.47	**58**	24	18	0
C4000.5	9.38	9.49	10.03	16.53	20.54	17.73	26	**53**	16	5
MANN_a45	18.22	14.16	18.71	18.45	22.78	20.03	0	**85**	0	15
MANN_a81	15.07	19.88	28.58	54.49	216.75	27.04	0	**95**	0	5
keller6	0.43	0.40	0.63	1.18	413.15	1.19	0	**97**	3	0
brock800_1	−−	8.94	5.18	4.01	2.81	2.63	0	0	1	**99**
brock800_2	−−	5.71	3.30	2.23	1.64	2.25	0	0	2	**98**
frb53-24-2	−−	−−	−−	−−	−−	291.86	2	2	**96**	0
frb56-25-4	−−	−−	−−	−−	−−	97.60	1	11	**88**	0

Table 6.5: Average run-times, over 100 successful trials for CLS and all possible combinations of parallel copies of RLS and PLS. An entry of −− means that the combination of runners is not able to solve the instance successfully on 100 runs. The last four columns denote the low level heuristic which actually solved the instance in the 100 runs.

sations. The comparison is performed on the most challenging DIMACS and BHOSLIB instances. The combination of RLS and PLS runners is stopped once one of the copies of the heuristics finds the optimum solution. It is clear from the table that no combination of single heuristics is able to outperform CLS on any of the selected instances.

The results presented in Table 6.5 can be explained looking at right-most columns of the table. For example, MANN_a81 is almost always solved by RLS or a LEVEL heuristic. After 1,000,000 selections, CLS re-configures to a Plateau configuration with 2.5 cores dedicated to the LEVEL heuristic; therefore, the performances should lie be between a RRPP and a RRRP combination. Since CLS has also the startup time before the re-configuration, the performances are more similar to RRPP than RRRP. On the BHOSLIB selected instances there is no combination of RLS and PLS which solves the problem in all 100 runs. The lack of the FOCUS heuristic, which is the most effective on these hard instances, is the reason for the poorer performances.

Figure 6.3 shows the average vertex degree distributions of the candidate solutions encountered by GREEDY, PENALTY, and FOCUS. The combined curve of GREEDY and PENALTY has two peaks corresponding to the bias towards low degree vertices of the penalty based heuristic and to the high

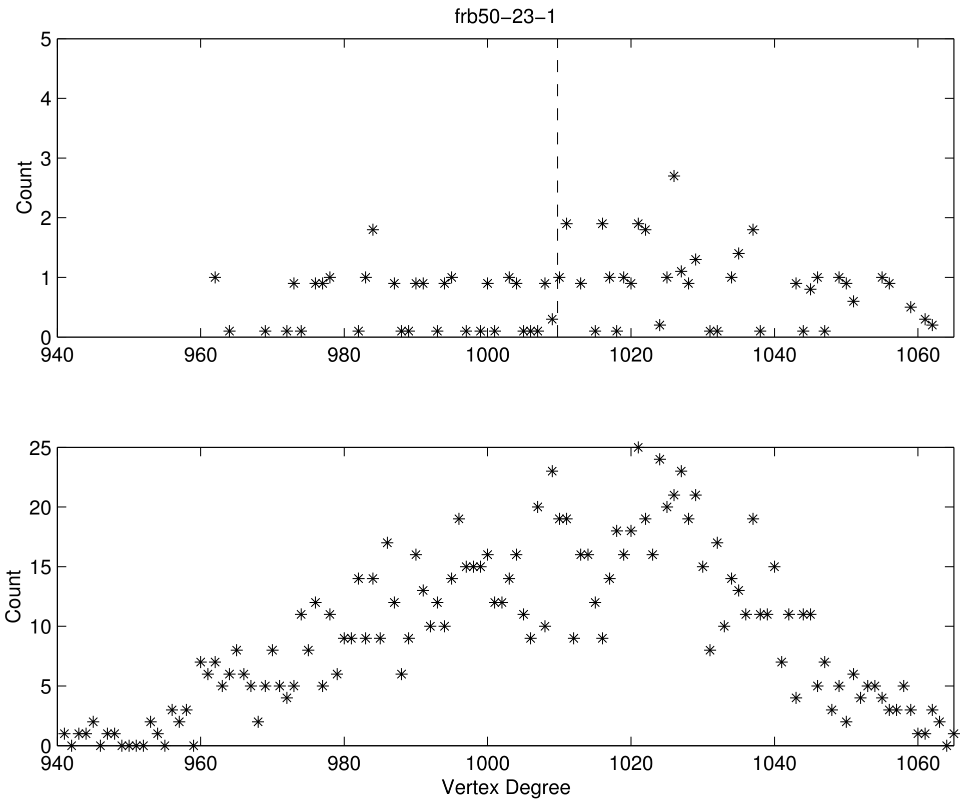


Figure 6.2: For frb50-23-1, the top graph shows, how frequently a vertex with a specific degree appeared in a maximal solution. The dashed line indicates the average vertex degree of the maximum clique. Minimum and maximum vertex degree are 941 and 1065 respectively. The distribution of the 1150 vertices is shown in the graph on the bottom.

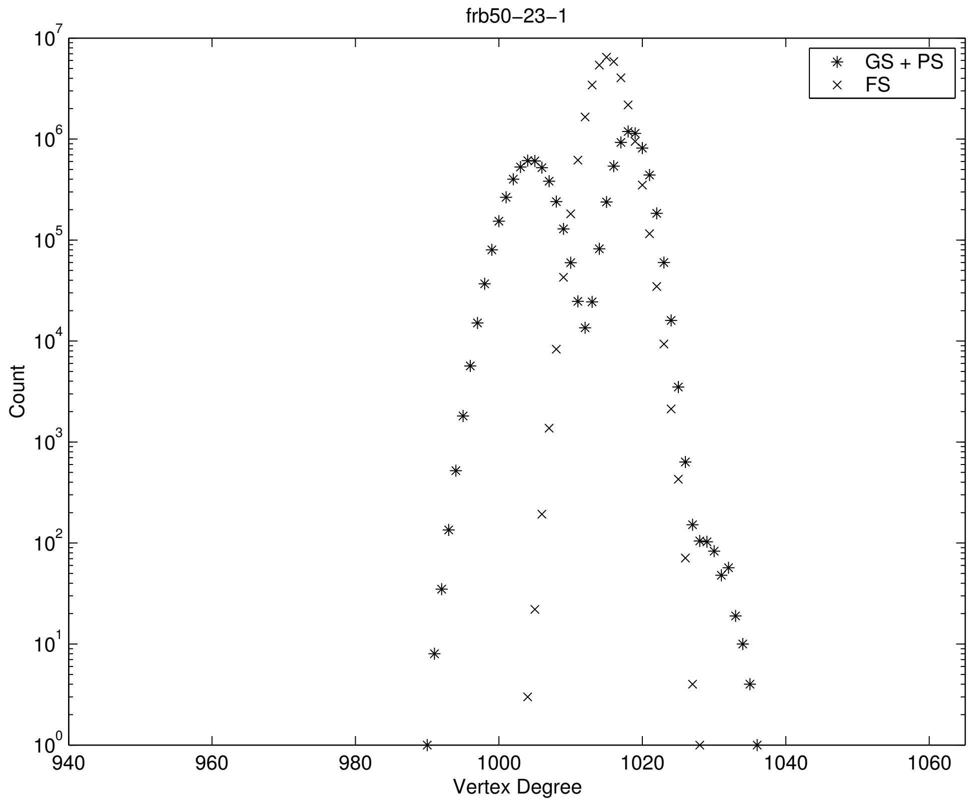


Figure 6.3: For frb50-23-1, the frequencies of the average vertex degree of maximal cliques encountered by GREEDY + PENALTY ($GS + PS$) and FOCUS.

Instance	Cores							
	1	2	3	4	5	6	7	8
MANN_a81	141.35	78.81	31.30	27.04	19.09	17.56	11.31	10.88
frb56-25-4	881.00	406.19	158.61	97.60	76.45	60.75	52.25	48.89

Table 6.6: CLS average run time in seconds averaged over 100 trials for two hard instances from the DIMACS and BHOSLIB benchmark datasets.

degree vertices of the greedy heuristic. Using the information received from GREEDY and PENALTY, the FOCUS heuristic is able to fill the gap between the two peaks. The average degree of the optimum solution for this instance (see Figure 6.2) lies exactly in this gap.

Table 6.6 shows the speedup of CLS on two representative instances of the DIMACS and BHOSLIB benchmarks. These two instances have been chosen because they are hard instances and represent the two extreme cases: one is tackled effectively by a Plateau configuration (MANN_a81) while the other (frb56-25-4) by a Non-Plateau configuration. The table shows how the performance of CLS improves when more cores are available. In the case of the MANN_a81 instance, the LEVEL heuristic which is essential for effectively solving the instance, goes from one quarter of a CPU core in the single core configuration up to 6.5 cores in the eight cores configuration. Conversely in the case of the frb56-25-4 where the fundamental heuristic is FOCUS a Non-Plateau configurations allocates one quarter of a core in the single core version up to 6.5 cores in the eight cores version.

6.5 Conclusions

This chapter presented CLS, a new hyper-heuristic for the MC problem that improves over the state-of-the-art gaining unprecedented robustness and consistency on the DIMACS and BHOSLIB benchmark datasets.

CLS combines in parallel four low level heuristics which are effective on a heterogeneous set of instances. Moreover communication between the low level heuristics allows to have truly complementary heuristics that focus on different parts of the search space. The first heuristic, GREEDY (based on PLS greedy phase), uses the vertex degrees to guide the search towards candidate solutions having a high average vertex degree. The second heuristic, LEVEL (based on RLS–LTM), is characterised by extensive plateau searches and uses a reactive tabu mechanism to avoid cycling through the same configurations. Also LEVEL uses the vertex degree to guide the search towards candidate solutions having a high vertex degree. The PENALTY heuristics (based on PLS penalty phase) uses the penalty values collected by GREEDY to bias the search towards solutions having a low average vertex degree. Information on the average vertex degree of the maximal configurations visited by GREEDY and PENALTY is regularly passed to FOCUS: a fourth low level heuristic which explores areas of the search space that GREEDY and PENALTY tend to ignore. In addition, after an initial explorative phase, CLS dynamically reallocates copies of the four heuristics to CPU cores, in order to ensure that the most effective mix of low level heuristics for the instance at hand is used. The decision of the most appropriate mix is driven by the analysis of the size of plateau areas performed by GREEDY during the explorative phase.

CLS performance is comparable and sometimes improves over single heuristic optimised for specific instances of the DIMACS benchmark dataset. Moreover the overall robustness and consistency on both DIMACS and BHOSLIB instances clearly demonstrates the effectiveness of the underlying paradigm of combining dynamic local search heuristics. The technique can also provide a basis for the design of novel hyper-heuristics for weighted maximum cliques and other related optimisation problems.

Chapter 7

Supervised Clustering

As an application of the study described in the previous chapters, we tackle the problem of predicting the set of residues of a protein that are involved into the binding of metal ions and more generally participating in active sites. We propose in this chapter a supervised clustering method [46] that achieves substantial improvements over a previous structured-output approach for metal binding site prediction. Significant improvements over the current state-of-the-art are also achieved in predicting catalytic sites from 3D structure in enzymes.

7.1 Introduction

In order to accomplish their biological function, proteins often interact with different types of external molecules such as metal ions, prosthetic groups and various organic compounds. Metalloproteins [15] bind metal ions in order to stabilise their three-dimensional structure, induce conformational changes or assist protein function, such as electron transfer in cytochromes. Metal binding sites are characterised by the set of protein atoms directly involved in binding the ion, called ligands, and the overall geometry of the site. Furthermore, the same protein often binds multiple ions, with typical numbers ranging from one to four. Enzymes are a fundamental type of proteins which accelerate chemical processes within a cell, by complexing with the substrate and thus lowering the activation energy of the reaction. Functional residues play various roles in the catalytic process, such as donating electrons or polarising cofactor bonds [6]. Solely binding substrates, cofactors or metals, which are often involved in enzymatic reactions, does not characterise a residue as catalytic according to the Catalytic Site Atlas (CSA) [52].

Being able to predict metal binding sites as well as enzyme active sites in novel proteins is a fundamental step in understanding their functioning. Both problems have been mostly addressed as a binary classification task at the residue level: given a protein sequence, predict for each residue whether it is involved in a metal binding site [49], [57] or an active site [61], [24] respectively. Most existing approaches for modelling the full metal binding geometry assume knowledge of the 3D structure of the protein [27, 3] and focus on detecting apo-proteins, i.e. proteins solved without the ion. A recent attempt [29] to predict metal binding geometry from sequence formulates the problem as a structured-output task. The proposed solution is a search algorithm greedily assigning residues to ions (or a default *nil* ion if predicted as free) guided by a scoring function trained to rank correct moves higher than incorrect ones. The algorithm is guaranteed to find the solution maximising the overall score, given the matroid structure of the problem. However, the scoring function is learned from examples and there is no guarantee that it correctly approximates the true underlying function.

We take here a different viewpoint and formalise the problem as a distance-based supervised clustering task [7]. Given a set of training instances, we first learn a similarity function predicting whether two residues jointly participate in a certain metal or active site. The learned similarity measure is subsequently fed to a maximum-weight clique algorithm collecting sets of residues maximising their pairwise similarities. The algorithm has a number of desirable features including automatic selection of the number of clusters, natural handling of overlapping clusters, and scalability to large datasets. Experimental results show a substantial improvement over the structured-output approach for metal binding geometry prediction. Significant improvements over the state-of-the-art are also obtained for active site prediction from protein 3D structure, where both node and edge weights are employed in order to exploit both local predictions and spatial constraints.

7.2 Problem Description and Formalisation

As already formalised in Chapter 1, let us consider a protein as a string in Σ^* over a small alphabet Σ. We want to learn a function that given a string $s \in \Sigma^*$ maps it to a partial clustering $\mathcal{Y}$, i.e., a set of disjoint subsets of the positions in s, where each subset contains the elements in the string that belong to the same active site, or that cooperate in the binding of a metal ion. More formally, if the length of the string is l, we define the set of all possible partial clusterings of the l indices in the following way:

$$\mathcal{C}_l = \Big\{ \mathcal{Y} \subseteq \mathcal{P}(\{1, \ldots, l\}) : \bigcup \mathcal{Y} \subseteq \{1, \ldots, l\} \wedge \\ (A, B \in \mathcal{Y}\ A \neq B \Rightarrow A \cap B = \emptyset) \wedge \emptyset \notin \mathcal{Y} \Big\}.$$

The set $\mathcal{C}_l$ is finite since $\mathcal{C}_l \subseteq \mathcal{P}(\mathcal{P}(\{1, \ldots, l\}))$. We define the set $\mathcal{C}$ as:

$$\mathcal{C} = \bigcup_{l \in \mathbb{N}} \mathcal{C}_l.$$

Given an example set of known mappings:

$$S \subseteq \Sigma^* \times \mathcal{C},$$

such that each string in the example set S has a mapping to only one clustering $\mathcal{Y}$:

$$[(s,\mathcal{Y}),(s',\mathcal{Y}') \in S \wedge \mathcal{Y} \neq \mathcal{Y}'] \Rightarrow s \neq s', \qquad (s,\mathcal{Y}) \in S \Rightarrow \mathcal{Y} \in \mathcal{C}_{|s|},$$

we want to learn a function:

$$f : \Sigma^* \to \mathcal{C}, \tag{7.1}$$

that extends the map defined by S:

$$(s,\mathcal{Y}) \in S \Rightarrow f(s) = \mathcal{Y}, \qquad f(s) \in \mathcal{C}_{|s|}.$$

We split the problem of predicting with structured output in two parts: first we learn a pairwise similarity measure on the elements in each cluster with a binary classifier, then we use it to construct a edge-weighted graph and mine the clusters as weighted MC. We use as positive examples all pairs of elements which belong to the same cluster, and as negative examples all other pairs, obtaining the training set S':

$$S' \subseteq \Sigma^* \times \mathbb{N} \times \mathbb{N} \times \{\pm 1\},$$

where

$$\begin{gathered}(s,u,v,y) \in S' \Leftrightarrow \\ u,v \in \{1,\dots,|s|\} \\ \wedge \exists \mathcal{Y} \in \mathcal{C}_{|s|} : [(s,\mathcal{Y}) \in S \wedge (y = +1 \Leftrightarrow u \neq v \wedge \exists C \in \mathcal{Y}\ \{u,v\} \in C)].\end{gathered}$$

We use S' to train a support vector machine SVM : $\Sigma^* \times \mathbb{N} \times \mathbb{N} \to \mathbb{R}$ that given a string s and pair u,v returns a score that is a confidence that u,v belong to the same cluster. In our case the score is the distance from the margin of the trained classifier.

Given a string $s \in \Sigma^*$, we use the trained SVM margin function to build a weighted graph $G^s = (V^s, E^s, F^s)$, where $V^s = \{1,\dots,|s|\}$, $F^s(u,v) =$

$t(\mathrm{SVM}(s, u, v))$ with t being a scaling function in a suitable range, and $E^s(e^s_{u,v})$ being the adjacency matrix where:

$$e^s_{u,v} = \begin{cases} 1 & \text{if } \mathrm{SVM}(s, u, v) \geq \theta \\ 0 & \text{otherwise} \end{cases}$$

and θ is a suitable threshold value. By construction, it is clear that cliques with the highest weight in G^s correspond to the desired clusters in s. The threshold θ accounts for errors in the prediction of the weights on the edges.

Back to the biological details, given a protein sequence as a string of characters in the alphabet of 20 amino acids, the problem consists of: detecting the number of binding or catalytic sites; collecting for each site the set of protein residues involved. Metal binding sites tend to be rather specific in terms of possible ligands with cysteine (C), histidine (H), aspartic (D) and glutamic (E) acids being by far the most common ligands in transition metals. Cysteines and histidines are the vast majority of ligands in structural sites, while aspartic and glutamic acids are quite common in proteins and their relative binding frequency is thus very limited [49]. A more complex situation can be observed with alkali and alkaline-earth metals, which often bind proteins through the oxygen in backbone carbonyl groups. Catalytic propensity is even less specific, given the number of different roles that a residue can play within the active site. Figure 7.1 reports the catalytic propensity of the whole set of amino acids, showing that only few of them can be safely discarded. Previous results [24] on the simpler binary classification task actually indicate that keeping all candidates produces slightly better results on average: the predictor occasionally manages to correctly predict rare amino acids as catalytic without significantly affecting precision.

Concerning the number of sites, metalloproteins usually contain between one and three sites, sometimes four and occasionally more. The coordina-

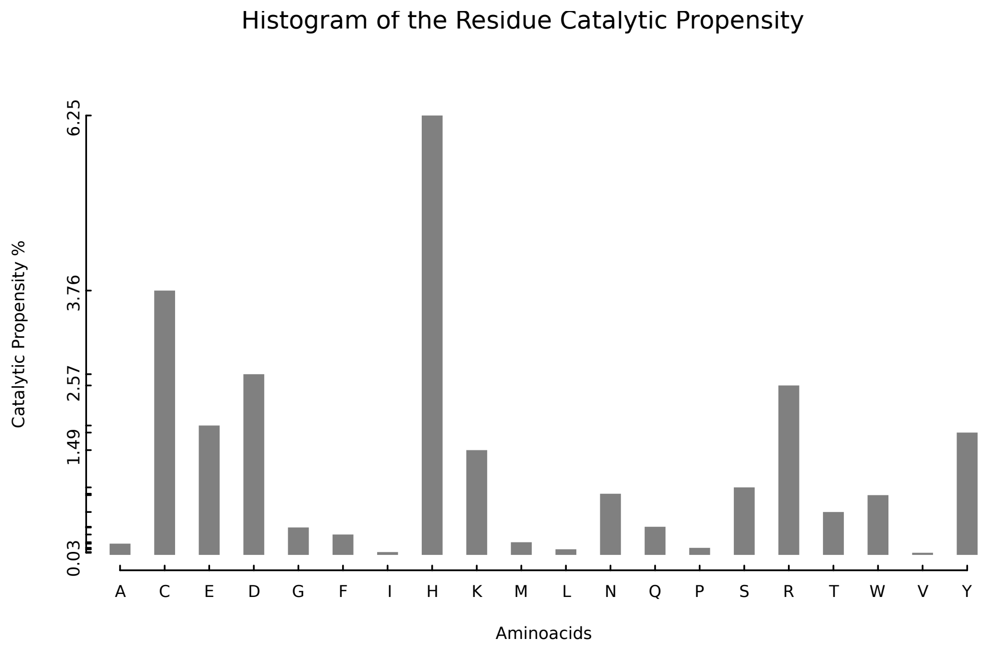


Figure 7.1: Histogram of the catalytic propensities of the residues in the experimental dataset *HA superfamily* (see experimental section for details).

tion number of a bound ion, i.e. the total number of its ligands, varies from one to about eight depending on the metal. Values between two and four are the most frequent for transition metals. Figure 7.2 shows the metal binding geometry of the equine herpes virus-1 (PDB code 1CHC), where candidate ligands in $\mathcal{L} = \{C, H\}$ not binding any ion are marked in lightest shade of grey. Contrarily total metal binding sites, enzymes tend to have a single catalytic site involving a larger number of residues, ranging from 1 to 9 in the experimental dataset we used. Multiple active sites can actually be found in some multimeric proteins, such as the 3-isopropylmalate dehydrogenase (PDB code 1A05). Figure 7.3 shows the active site of cloroperoxidase T (PDB code 1A7U and UniProtKB entry O31168) with seven residues corresponding to seven different amino acids involved. Note that proximity in sequence only partially relates to involvement in the same site, as the three-dimensional arrangement of the protein

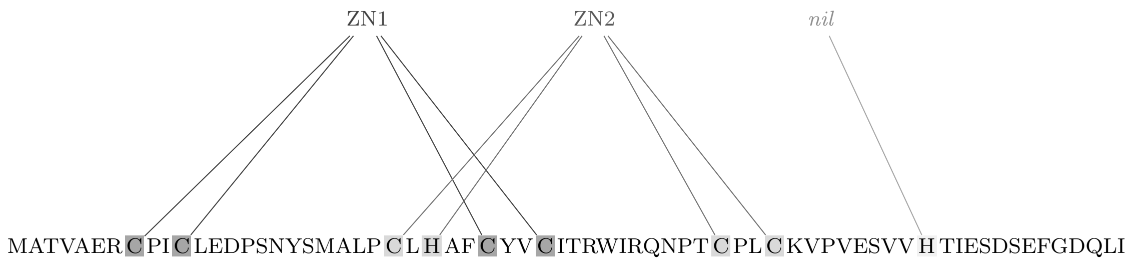

ZN1
ZN2
nil
MATVAERCPICLEDPSNYSMALPCLHAFCYVCITRWIRQNPTCPLCKVPVESVVHTIESDSEFGDQLI

Figure 7.2: Sequence of the equine herpes virus-1 (PDB code 1CHC). Residues composing the metal binding sites are highlighted in different shades of grey.

MPFITVGQEN STSIDLYYED HGAGQPVVLI HG**F**PLSGHSW 40
ERQSAALLDA GYRVITYDRR GFGQSSQPTT GYDYDTFAAD 80
LNTVLETLDL QDAVLVGF**SM** GTGEVARYVS SYGTARIAKV 120
AFLASLEPFL LKTDDNPDGA **AP**KEFFDGIV AAVKADRYAF 160
YTGFFNDFYN LDENLGTRIS EEAVRNSWNT AASGGFFAAA 200
AAPTTWYTDF RADIPRIDVP ALILHGTG**D**R TLPIENTARV 240
FHKALPSAEY VEVEGAP**H**GL LWTHAEEVNT ALLAFLAK

Figure 7.3: Sequence of the cloroperoxidase T (PDB code 1A7U and UniProtKB entry O31168). Residues composing the active site are highlighted in bold.

can bring quite distant residues closer. However, additional features contribute to characterise target residues, such as conservation profile and residue neighbourhood.

Given these premises, we formulate the problem as a supervised clustering task. We provide a common formulation for both metal binding site and active site prediction. Slightly abusing terminology, we refer to residues involved in either type of site as *ligands*. While the two problems are treated as separate tasks in the experiments, they are indeed highly correlated as metal binding sites are often part of a larger active site. We are planning to extend our work to predict a structured set of sites in order to jointly address these problems.

A protein sequence is represented as the set x of its candidate ligands, that is residues belonging to $\mathcal{L}$. The output y for the sequence is a subset of the powerset of x, i.e. $y \subseteq \mathcal{P}(x)$. Outputs for proteins in Figures 7.2 and 7.3, for instance, would be represented as $\{\{c_1, c_2, c_4, c_5\}, \{c_3, h_1, c_6, c_7\}\}$ and $\{f_2, s_8, m_2, a_{14}, p_7, d_{18}, h_6\}$ respectively, assuming $\mathcal{L}$ is equal to $\{C, H\}$ for metal binding sites and the whole set of amino acids for catalytic sites. The desired output is thus a partial clustering of residues, where only predicted ligands are reported. Furthermore, at least for metal binding sites, clusters can overlap, as the same residue can simultaneously bind two ions, as happens for glutamic and aspartic acids with their two side-chain oxygen atoms. For comparison with previous approaches, experiments only deal with non-overlapping clusters, but our approach can naturally handle overlaps, as described in the next section.

7.3 Distance-based Clustering with Maximum-weight Cliques

A training set of labelled proteins can be easily obtained from experimentally solved protein structures and catalytic annotations, and a supervised clustering approach can thus be pursued. We opt for a distance-based supervised approach [7], where training instances are used to learn an appropriate distance (or similarity) measure to be later used in the clustering. The learning stage simply consists of training a pairwise classification function $F(x^i, x^j)$ predicting for each pair of residues x^i and x^j in x whether they belong to the same site. We employ a pairwise support vector machine (SVM) as the underlying classification function. More complex alternatives can be pursued, as will be detailed in the Discussion.

Given a learned similarity function F, we represent a set x as a weighted graph, removing edges whose weight is below a certain threshold θ and rescaling remaining weights to be positive. A maximum-weight clique algorithm is then run on the graph in order to return a set of maximal cliques, which correspond to the predicted sites. The rationale for the approach is that given a reasonable pairwise similarity measure, the algorithm should isolate few densely connected components which correspond to the desired solution while discarding most of the nodes in the graph. The algorithm can be asked to return a single large cluster, as typical of the active site prediction task, or a set of possibly overlapping maximal cliques, as for the metal binding site case, where the number of clusters cannot be specified *a priori*.

7.4 The Maximum-weight Clique Algorithm

In the following we introduce our heuristic algorithm. We describe it for weighted edges only. Its extension for dealing with weights on both nodes and edges, as well as the case where weights are averaged on the number of nodes, is straightforward.

Given a set of residues R, in the previous section we defined a learned symmetric similarity function F that maps each pair of residues onto a measure of likelihood that they belong to the same cluster. Given a positive threshold value θ, we define a weighted undirected graph as a triplet $G_\theta \equiv (R, E_\theta, F)$ where the vertex set R is composed by the residues, the edge set E_θ is defined by vertex pairs whose similarity function F is above the threshold θ

$$E_\theta = \big\{\{u, v\} \subset R : u \neq v \wedge F(u, v) \geq \theta\big\},$$

and the weight of every edge $e \in E_\theta$ is given by $F(e)$. From now on, subscript θ shall be removed for clarity.

The *Edge-Weighted Maximum Clique Problem* requires to find the clique in R that maximises the sum of weights:

$$R'_{\max} = \mathop{\arg\max}_{\substack{R' \subseteq R \\ R' \text{ clique in } G}} \sum_{u,v \in R'} F(u,v).$$

Being a generalisation of the Maximum Clique Problem, the edge-weighted version is also NP-hard. In this chapter, we introduce the Reactive Local Search optimisation heuristic for Weighted Maximum Clique finding (RLS-WMC, in the following WMC for short), based on the RLS–MC heuristic for Maximum Clique finding [13], with a novel dynamic behaviour adapted from the one described in Chapter 4.

The reaction technique of the WMC heuristic, described below, offers an effective diversification mechanism that provides a thorough exploration of the search space, and is therefore capable of dealing with problem instances for which exhaustive enumeration is infeasible.

The WMC heuristic, whose main section is shown in Listing 7.1, is a stochastic local search (SLS) algorithm. In SLS algorithms for the MC problem, a 'current' configuration (subset of vertices) $\bar{R} \subseteq R$ is maintained throughout the search, being initially the empty set (line 4), and is modified by incremental moves consisting in the addition or in the removal of a node (lines 8–11). At every step the 'current' configuration is required to be a clique in the original graph (the system generally moves only within feasible solutions), therefore the addition move will only consider nodes that maintain the clique property, i.e., that are connected to all nodes in $\bar{R}$. Such set of eligible nodes is called P in Listing 7.1, and is maintained incrementally during the search.

The WMC heuristic completes the generic SLS framework by defining the criteria by which the incremental moves are selected. In particular, a parameter T, called *prohibition period*, is set and a vector $(L_v)_{v\in R}$, storing the last iteration at which node v was added or removed to the current clique $\bar{R}$, is initialised (line 4) and maintained (line 13). Nodes that have

Listing 7.1: Main section of WMC; bookkeeping operations such as best configuration maintenance are not shown.

Input	Meaning
R, E, F_E	Edge-weighted undirected graph

Variable	Meaning
t	Current iteration index
T	Prohibition period
L_v	Last iteration when $v \in R$ was added/removed
$\bar{R}$	Current configuration
P	List of nodes that can be added to $\bar{R}$
w	Clique weight
v	Chosen node
a	Action to be taken (`Add` or `Drop`)

```
function WMC($R, E, F_E$)
    $L_v \leftarrow -\infty$ for $v \in R$
    $t \leftarrow 0; \bar{R} \leftarrow \emptyset; P \leftarrow R; w \leftarrow 0$
    repeat
        UpdateProhibition($\bar{R}, T$)
        $(v, a) \leftarrow$ ChooseNode($L, \bar{R}, P, T, t, E, F_E$)
        if $a =$ Add
            $\bar{R} \leftarrow \bar{R} \cup \{v\}$
        else
            $\bar{R} \leftarrow \bar{R} \setminus \{v\}$
        recompute $P$ and $w$ incrementally
        $L_v \leftarrow t$
        if too many iterations without improvements
            Restart()
        $t \leftarrow t + 1$
    until termination condition is met
    return best $\bar{R}$ found
```

been used in the last T iterations, called *prohibited*, are not considered for addition or removal. This mechanism, known as *Tabu Search*, prevents the system from getting stuck in local optima and encourages diversification.

The move selection routine ChooseNode, whose purpose is the choice of the next node to be added or removed, is outlined in Listing 7.2. Array (L_v) is used to check prohibitions. Since more than one non-prohibited node is usually eligible for addition to $\bar{R}$, other selection criteria intervene in order to maximise the chance that a large clique will be obtained, for instance by choosing the node that maximises the average edge weight (line 3), with ties broken randomly (line 8). If no nodes are eligible for insertion in the current configuration $\bar{R}$ (either because there are no more nodes connected to all nodes in $\bar{R}$, or all of them are prohibited), then a non-prohibited node chosen within $\bar{R}$ is selected for removal (lines 5–7).

The value of the prohibition period T is critical for the good behaviour of the algorithm. Small values of T tend to be insufficient for the system to efficiently escape local optima, while high values highly reduce the flexibility of the search procedure by reducing the number of eligible nodes. Rather than relying on an ideal value of T as a function of the graph size and of its density, WMC determines it dynamically (line 6 of Listing 7.1) by calling a function, UpdateProhibition, that detects anomalous situations where a change would benefit the search. To achieve this, recent configurations are stored in a hash table; if a configuration is visited (i.e., becomes the current one) too often, then the T parameter is increased in order to improve the differentiation capabilities of the algorithm. If, on the other hand, no configuration is revisited for a given time, T is reduced. Further details on the dynamic adaptation of T have been described in Chapter 5.

Finally, a Restart mechanism is provided (lines 14–15): if the best solution is not improved in a while, then the algorithm is restarted, so that

Listing 7.2: ChooseNode chooses the non-prohibited node having the best chance to lead to better cliques in the future; if no nodes can be added, it picks one for removal.

Input	Meaning
L_v	Last iteration when node v was added/removed
$\bar{R}$	Current configuration
P	List of nodes that can be added to $\bar{R}$
T	Prohibition period
t	Current iteration index
E, F_E	Edges and weights

Variable	Meaning
S	Set of nodes eligible for adding or removing

Output	Meaning
v	Chosen node
a	Action to be taken (`Add` or `Drop`)

```
function ChooseNode($L, \bar{R}, P, T, t, E, F_E$)
    $S \leftarrow \left\{ w \in P : \begin{matrix} L_w > t - T \wedge \\ \wedge\, w \text{ maximises future expectations} \end{matrix} \right\}$
    $a \leftarrow$ Add
    if $S = \emptyset$
        $S \leftarrow \left\{ w \in \bar{R} : \begin{matrix} L_i > t - T \wedge \\ \wedge\, w \text{ maximises future expectations} \end{matrix} \right\}$
        $a \leftarrow$ Drop
    Pick $v \in S$
    return $(v, a)$
```

new regions of the search space are visited. The RLS-WMC algorithm maintains the weight of the current configuration $\bar{R}$ by incrementally updating it at every move.

For the purposes of this chapter, cliques within the expected size are stored along with their weight, and are post-processed in order to determine which ones represent the correct clusters. Bookkeeping operations such as the computation of the clique weight, storage of the visited cliques and of the best clique are not detailed here.

7.5 Experimental results

7.5.1 Predicting geometry of metal binding sites

We tested our method on the task of predicting metal binding sites in metalloproteins. We used the same setting described in [29], with 30 random 80/20 train/test splits. We encoded pairs of residues by concatenating their features vectors, thus comparing residues according to their order in the sequence. This option was shown [29] to provide better results with respect to alternative approaches such as averaged pairwise comparisons, possibly because sequential ordering is relevant in characterising sites. Pairs were labeled positive if both residues bind to the same metal ion and negative otherwise, and an SVM was used as the pairwise classifier.

All parameters concerning the SVM and the maximum weighted clique algorithm described below were selected by an inner-fold cross-validation on the training set of the first split and kept fixed for all remaining folds. As a result of this model selection phase, we employed a second degree polynomial kernel and a cost factor $j = 3$ outweighing error on positive with respect to negative examples. In building the weighted graph, we discarded edges having weight smaller than -0.9, and rescaled remaining weights to have positive values. Since the maximum-clique algorithms accepts only positive integer weights, we scaled the margins mapping them to unsigned integers in the following way: $m' = \lfloor m * 10^p \rfloor - \theta$, where m is the margin outputted by the SVM, $p = 3$ is the number of decimals to be considered, and θ is a threshold that allows to embed also negative margins (up to -0.9) in the graph. The θ parameter has been learned in the inner-fold cross-validation. The graph is not completely connected, and can actually be very sparse, since edges for negative margins m' are not reported in the graph. The weight of each clique was averaged over the number of its nodes. Now, the maximum clique algorithm, enumerates all cliques of size up to 4, which average weight is maximal. The choice of the model describing the most relatively heavy solution, has also been performed in the inner-fold cross-validation. The model used in the experiments averages the overall weight of a configuration $\bar{R}$ by dividing it by its size:

$$F_E(\bar{R}) = \frac{\sum_{\{u,v\}\in\bar{R}} F_E(\{u,v\})}{|\bar{R}|}.$$

By doing so, the algorithm outputs not the heaviest cliques, but the heaviest relative to the their size. These solutions are maximal, which means that any other possible node added would decrement the average weight of the solution. For example, among the alternative models, one could also average the solution by dividing the total weight by the number of edges:

$$F_E(\bar{R}) = \frac{\sum_{\{u,v\}\in\bar{R}} F_E(\{u,v\})}{\begin{pmatrix} |\bar{R}| \\ 2 \end{pmatrix}};$$

but this would give a negative bias to bigger cliques. The algorithm returned the set of non-overlapping solutions with at most four residues. We made no further selection of the returned solutions, except for limiting the number of solutions to 4. We present here a set of measures including those reported in [29]. Note that we are not trying to predict the identity

# sites	SVM + WMC			[29]		
	P_E	R_E	F_E	P_E	R_E	F_E
any	79 ± 3●	59 ± 5●	62 ± 5●	66 ± 5	52 ± 4	53 ± 4
1	84 ± 4	73 ± 7	73 ± 6	66 ± 7	58 ± 6	57 ± 6
2	70 ± 8	33 ± 5	42 ± 6	67 ± 7	44 ± 9	48 ± 9
3	70 ± 15	22 ± 8	32 ± 11	69 ± 19	24 ± 13	32 ± 12
4	42 ± 30	16 ± 13	23 ± 18	42 ± 31	20 ± 19	26 ± 22
	P_S	R_S	F_S	P_S	R_S	F_S
any	42 ± 7●	30 ± 7●	31 ± 7●	20 ± 7	17 ± 6	16 ± 6
1	50 ± 8	41 ± 9	41 ± 9	25 ± 10	22 ± 8	22 ± 8
2	25 ± 14	8 ± 7	11 ± 9	15 ± 9	7 ± 7	7 ± 7
3	23 ± 32	4 ± 7	5 ± 11	0 ± 2	0 ± 1	0 ± 2
4	9 ± 21	3 ± 6	5 ± 9	2 ± 7	1 ± 5	1 ± 5
	P_B	R_B	F_B	P_B	R_B	F_B
any	88 ± 3●	63 ± 5	67 ± 4●	79 ± 4	64 ± 6	64 ± 4
1	84 ± 4	73 ± 7	73 ± 6	74 ± 5	68 ± 7	65 ± 6
2	92 ± 8	45 ± 6	58 ± 7	88 ± 5	60 ± 11	66 ± 10
3	100 ± 0	34 ± 12	49 ± 15	98 ± 5	38 ± 22	50 ± 20
4	67 ± 45	25 ± 18	36 ± 25	65 ± 44	32 ± 28	40 ± 31

Table 7.1: Comparison on the metalloproteins dataset. The means and standard deviations are computed on the 30 random splits. A bullet indicates that the performance differences are statistically significant ($p < 0.05$).

of an ion (e.g. the 'first' zinc, the 'second' iron or so), but only the subset of residues which jointly bind the same one. Thus, when evaluating the quality of a certain clustering, we assign each ion to the cluster containing the highest number of its true ligands (if any). An equivalent approach was employed in [29]. P_E, R_E, and F_E are the precision, recall, and F_1 of the correct assignment between a ligand and a metal ion. P_S, R_S, and F_S are the precision, recall, and F_1 of the correct prediction of binding sites, i.e., how many sites are entirely correctly predicted over the total number of sites in the chain. P_B, R_B, and F_B are the precision, recall, and F_1 of the correct prediction of the bonding state of the residues in the chain, i.e. regardless of which ion they actually bind. Tables 7.1 and 7.2 report the mean and standard deviation of these performance measures averaged over the 30 splits. The breakdown of these measures for proteins binding different numbers of metal ions (i.e. from 1 to 4) is also reported.

Our SVM+WMC approach achieves significant improvements over the previous structured-output approach in edge, site and bonding state prediction, as measured by paired Wilcoxon tests ($p < 0.05$).

	# sites				
	any	1	2	3	4
SVM + WMC	27 ± 6●	40 ± 9	1 ± 4	0 ± 0	0 ± 0
[29]	14 ± 6	20 ± 8	3 ± 7	0 ± 0	0 ± 0

Table 7.2: Experimental results on the metalloproteins dataset. A_G is the accuracy at a chain level, i.e., the number of entire configurations correctly predicted. A bullet indicates that the performance differences are statistically significant ($p < 0.05$).

The most significative improvement over [29] lies in the number of sites entirely correctly predicted. The overall P_S, R_S, and F_S, is consistently better for any number of metal ions in the protein.

7.5.2 Active sites prediction

We applied our approach to the prediction of active sites in enzymes. We focused on the *HA superfamily* dataset [22], the largest dataset employed as benchmark in the literature. Prediction of catalytic residues was previously addressed starting from either sequence or structural information. We considered both settings, relying on previous state-of-the-art results by a simple support vector machine exploiting residue structural neighbourhood [24]. The detailed description of the features employed for both sequence-based and structure-based predictions can be found in this previous work. Given that most proteins contain a single active site, and the labelling found in the CSA [52] does not include information on different sites, we considered a single site prediction setting. Common examples of multiple active sites are those of polymeric proteins in which a pair of specular sites is found at the interface of two identical chains. We plan to extract this additional information from known 3D structures in order to fully characterise overall geometry in an extended version of the work.

For sequence-based prediction, we employed a setting analogous to the metal binding site case, with pairs of residues represented as ordered pairs of feature vectors from [24]. Following [24], we employed a linear kernel

	[24]			SVM+WMC		
	P	R	F_1	P	R	F_1
seq.	20 ± 4	59 ± 7	25 ± 4	22 ± 2	41 ± 4	27 ± 3 •
struct.	23 ± 3	65 ± 6	28 ± 3	35 ± 7	43 ± 7	34 ± 6 •

Table 7.3: Comparison of the results (performance ± st.d.) obtained in active site prediction. A bullet indicates that the performance differences are statistically significant ($p < 0.05$).

and a 6 to 1 subsampling of negative (i.e. non-catalytic) residues, resulting in a 61/1 proportion of negative vs. positive residue pairs. Following the site size distribution in training instances, we fixed the maximum size of cliques to six.

For structure-based prediction, we took a slightly different approach, since we could also exploit the spatial information provided by the protein structure. We modified the maximum-weight clique algorithm in order to consider both edge and node weights. Edge weights were in this case inverse Euclidean distances between corresponding residues, pruned for distances over 14 Å. This threshold was chosen according to the distribution of distances between catalytic residues in the training set. The idea of constraining candidate solutions based on their pairwise 3D distances was actually used in the MBG prediction approach by Babor et al. [3] as an initial filtering stage. However the 3D constraint is much less stringent in catalytic sites, as shown by the quite large threshold (14 Å) we derived from data. Node weights encoded catalytic propensity as predicted by the state-of-the-art support vector machine predictor described in [24]. Node and edge weights were normalised in order to fall within the same range of values.

Experimental comparisons with the local approach in [24] are shown in Table 7.3, where the protein-level precision, recall and F_1 measures averaged across folds are reported.

The SVM+WMC approach achieves significant improvements at $p < 0.05$ in both sequence-based and structure-based predictions according to a paired Wilcoxon test. Note that the average protein-level F_1 of the local predictor is quite lower than the F_1 computed from average protein-level precision and recall. This happens because the local SVM produces rather unbalanced predictions, either maximising recall with low precision or (more rarely) vice versa, and for a number of proteins it outputs completely wrong predictions. The SVM+WMC approach is much more stable and balanced in its predictions. Note also that the improvement in F_1 is not simply due to a better choice of the decision threshold with respect to the standard local approach. The best F_1 value which could be obtained with local sequence-based predictions by optimising the threshold (on the test set) is just 0.256. Results from the structured-based prediction significantly improve the current state-of-the-art thanks to an effective use of the spatial geometry information. In particular, the algorithm finds cliques that discard many of the classifier false positives.

7.6 Conclusions

We address the problem of predicting geometry of structural and functional sites in proteins by casting it into a supervised clustering task. We propose a novel distance-based supervised clustering approach in which the learned pairwise distance is employed to turn instances into weighted graphs. A maximum-weight clique algorithm is executed on the graph to return a small set of densely connected components corresponding to candidate sites. Supervised clustering is an active area of research and a number of different approaches have been proposed in the literature [7]. We use a very simple distance learning approach based on pairwise classification of instances. The maximum-weight clique clustering algorithm is however independent of this stage, and can be easily integrated in more complex supervised clustering approaches such as the structured-output formulation proposed in [28].

The algorithm substantially improves over the only existing approach in predicting geometry of metal binding sites from sequence alone. Focusing on small components with large overall weights, our algorithm is more robust to a possibly incorrect bonding state prediction. On the other hand, the structured-output approach in [29] is capable of exploiting the full relational structure of partial solutions in order to evaluate them, instead of being limited to networks of pairwise interactions. Indeed, such approach is superior when bonding state information is assumed to be known. We are planning to extend our algorithm in order to deal with clique-based weights, thus combining some of the advantages of the two formulations: the ability of a structured-output approach to better model the quality of candidate solutions, and the robustness of stochastic local search strategies in dealing with a scoring function which only approximates conditions guaranteeing greedy optimality [29].

Significant improvements over the state-of-the-art are also obtained in predicting active sites from 3D structure. The algorithm naturally handles the lack of knowledge in the number of clusters, partial clusterings with many outliers and overlapping clusters. We are planning to extend it to return a structured set of solutions, such as metal binding sites as parts of wider active sites, a quite common situation in enzymes.

Chapter 8

Conclusions

This thesis introduces analysis tools for improving the current state of the art of heuristics for the Maximum Clique (MC) problem. The analysis focussed on algorithmic building blocks of increasing complexity in order to understand their contribution in solving instances of the MC problem.

In Chapter 2, the experimental analysis on the two random graph classes show clearly that the plateau search is necessary to find the maximum clique in hard instances and in any case to reduce the average number of iterations. The complexity added to the algorithms by the plateau search does not increase the cost per iteration. On the contrary, especially for the algorithms using the dynamic degree for candidate selections, it reduces the CPU time per iteration. Moreover degree-based heuristics are more effective in more structured random instances, and are responsible for performance deterioration on pure random graphs. In general, the penalty heuristic is less robust than the prohibition heuristic, depending on the appropriate selection of the penalty value. Comparing more complex algorithms, RLS and RLS–StatDegree, always perform better then the other algorithms considered. The cost per iteration of RLS–StatDegree is bigger than the one of DLS–MC, although of the same order of magnitude. But the fewer steps needed on average to find the best cliques make

RLS the best choice for the two graph models considered in the analysis.

As a useful complement to the analytical work, in Chapter 3 we have presented a set of techniques for the visualisation of search landscapes which can support the researcher's intuition on the behaviour of a SLS algorithm applied to combinatorial optimisation problems. The visualisation also renders explicitly the geographic metaphors used by researchers to describe areas of interest of the landscape. The examples presented in this chapter are small instances useful to show how some features of the landscapes are rendered with the proposed techniques. The approximation techniques presented in Section 3.4 allow for the representation of instances otherwise intractable for the complete representation, while maintaining the features of the complete enumeration. Current research is aimed towards more scalable layout algorithms with no exogenous parameters that can lay out landscapes with more than few thousand solutions and tens of thousands of relations among them.

The analysis of the algorithmic building blocks described above tells just part of the story. A comprehensive work on algorithmic efficiency needs also considering low level implementation details, choices of data structures, programming languages, and knowledge on how the compiler optimises the code. The results of the investigation in Chapter 4 show that a careful implementation of the data-structures considering also operating system services like memory allocation achieves a significant reduction of the CPU time per iteration. The implementation of the supporting data structures of the new version has many improvements in the management of the dynamic memory and in the storage of the search history. Furthermore some algorithmic improvements to the original RLS have been introduced leading to the final RLS–LTM proposal. RLS–LTM achieves an order of magnitude difference in CPU times for graphs of reasonable sizes, and the difference appears to grow with the problem dimension. This results drastically changes the overall competitiveness of the Reactive Local Search technique.

In Chapter 5, a more focussed and extensive analysis of the Reactive Tabu Search (RTS) algorithm and of the datasets reveals that for almost every instance of the DIMACS benchmark set, there is a narrow range of good values that can be assumed by the prohibition parameter T of a Tabu Search heuristics. Outside this narrow range, the time needed by the heuristic to converge to good quality solutions increases significantly. The only notable exception are the MANN instances. A Robust Tabu Search centred around a correct setting for T performs as well as RLS. This result implies that at least on the DIMACS benchmark instances, there is no measurable effect showing that RLS effectively reacts to the local characteristic of the search space. The speedup in RLS–LTM can be ascribed to the faster restarts that do not need to clear the search history. Avoiding to clear the search history is one of the causes for the explosion of the tabu tenure parameter. Another cause is the bias in the node selection during the restarts. The fact that RLS–LTM also never resets the tabu tenure during the restarts makes the parameter explosion more noticeable. Knowing the reason for the speedup and for the explosion of the tabu tenure, we implement RLS–fast, which is as efficient as RLS–LTM and does not need for an upper bound for the tabu tenure. The performances of RLS–fast are comparable to RLS–fix an algorithm that knows a priori the best tabu tenure for every instance in the DIMACS benchmark. This result shows how RLS is able to quickly converge to the best tabu tenure for the instance at hand with very little overhead.

The analysis of the search dynamics and the improved RLS implementation, led naturally to the design of a new hyper-heuristic for the MC. In Chapter 6 we present Cooperating Local Search (CLS): a parallel hyper-heuristic that improves the state-of-the-art for the MC problem on the

DIMACS and BHOSLIB benchmark instances. CLS combines in parallel four low level heuristics which are effective on a heterogeneous set of instances. After an initial explorative phase, CLS dynamically reallocates copies of the four heuristics to CPU cores, in order to ensure that the most effective mix of low level heuristics for the instance at hand is used. Moreover, communication between the low level heuristics allows to have truly complementary heuristics that focus on different parts of the search space. CLS performance is comparable and sometimes improves over single heuristic optimised for specific instances of the DIMACS benchmark dataset. The overall robustness and consistency on both DIMACS and BHOSLIB instances clearly demonstrates the effectiveness of the underlying paradigm of combining dynamic local search heuristics. The technique can also provide a basis for the design of novel hyper-heuristics for weighted maximum cliques and other related optimisation problems.

As an application of the study presented in this thesis, in Chapter 7 we address the problem of predicting geometry of structural and functional sites in proteins by casting it into a supervised clustering task. We propose a novel distance-based supervised clustering approach in which the learned pairwise distance is employed to turn instances into weighted graphs. A maximum-weight clique algorithm is executed on the graph to return a small set of densely connected components corresponding to candidate sites. The algorithm substantially improves over the only existing approach in predicting geometry of metal binding sites from sequence alone. Focusing on small components with large overall weights, our algorithm is more robust to a possibly incorrect bonding state prediction. On the other hand, the structured-output approach in [29] is capable of exploiting the full relational structure of partial solutions in order to evaluate them, instead of being limited to networks of pairwise interactions. Significant improvements over the state-of-the-art are also obtained in predicting active sites from 3D structure. The algorithm naturally handles the lack of knowledge in the number of clusters, partial clusterings with many outliers and overlapping clusters. We are planning to extend it to return a structured set of solutions, such as metal binding sites as parts of wider active sites, a quite common situation in enzymes.

Although the work in this thesis has been very focussed and applied on a narrow class of algorithms for a single combinatorial optimisation problem, the insights gained from the analysis are much more general. Understanding the reasons behind the performances of single algorithmic build blocks helps not only in designing new algorithms for the problem at hand, but also for different, even unrelated problems. For example, knowing the bias of algorithmic components towards solutions having particular properties helps in designing hyper-heuristics or portfolio of techniques able to thoroughly explore the search space. The hyper-heuristic described in Chapter 6 goes exactly in this direction. In fact CLS is composed of low level heuristics that, thanks to information exchanges among them, can explore the search space looking for solutions with different average vertex degree, diversifying efficiently the search. Even more importantly the insight on the behaviour of the algorithmic building blocks is so important that, without it, it would be impossible to design the FOCUS heuristic that overcomes the limitation of the other building blocks, and is of extreme importance for solving particular classes of hard instances for the MC problem. The same can be said for the reallocation of heuristics to cores: without a proper knowledge of the extent and efficiency in exploring plateau areas, no informed decision about the characteristics of the instance, and consequently the best mix of heuristics to tackle it, could be taken. Researchers are becoming increasingly convinced that no single heuristic can solve efficiently all classes of instances of a combinatorial optimisation problem, therefore more effective solvers must have different algorithmic components to be selected and coordinated or generated for the instance at hand.

Understanding the role and performances of the building block helps in building better meta-heuristics and hyper-heuristics. But to design new and better building blocks one has to understand the reason behind their performance. While in Chapter 2 the performance of every building block was studied as a whole and the insights gained in such analysis applied in Chapter 4, in Chapter 5 we present a more fine-grained analysis. This latter analysis helps, for example, to understand the reason behind the tabu tenure explosion in RLS–LTM or the impact of a greedy selection of a seeding node in a restart on the overall search diversification. Having these insights could lead to the design of algorithms having the same performances of RLS without the burden of the complexity of managing a history of the configurations visited or, on the contrary, it could lead to the design of an algorithm that uses even more search history to make informed decisions on the amount of diversification in the restarts for the specific instance being optimised. There is no right or wrong answer, there are just fine-grained components that need to be carefully crafted and balanced together for the problem being optimised. This can be done automatically or manually, but in any case, in our opinion, it can not be done without the knowledge gained from in-depth analysis, aided by theoretical, experimental or even visual tools, of each algorithmic component.

Bibliography

[1] David Anderson, Emily Anderson, Neal Lesh, Joe Marks, Ken Perlin, David Ratajczak, and Kathy Ryall. Human-guided simple search: combining information visualization and heuristic search. In *Proceedings of the 1999 workshop on new paradigms in information visualization and manipulation in conjunction with the eighth ACM international conference on Information and knowledge management*, pages 21–25. ACM New York, NY, USA, 1999.

[2] David Pelta Antonio D. Masegosa, Franco Mascia and Mauro Brunato. Cooperative strategies and reactive search: A hybrid model proposal. In *Proceedings of Learning and Intelligent Optimization Third International Conference, LION 3, Trento, Italy, January 14-18, 2009*, pages 206–220, 2009.

[3] Mariana Babor, Sergey Gerzon, Barak Raveh, Vladimir Sobolev, and Marvin Edelman. Prediction of transition metal-binding sites from apo protein structures. *Proteins*, 70(1):208–217, 2008.

[4] Egon Balas and Chang Sung Yu. Finding a maximum clique in an arbitary graph. *SIAM Journal of Computing*, 15(4):1054–1068, 1986.

[5] Albert László Barabasi and Réka Albert. Emergence of scaling in random networks. *Science*, 286:509–512, October 1999.

[6] Gail J. Bartlett, Craig T. Porter, Neera Borkakoti, and Janet M. Thornton. Analysis of Catalytic Residues in Enzyme Active Sites. *J Mol Bio 2002*, 324(1):105–121, 2002.

[7] Sugato Basu. *Semi-supervised clustering: probabilistic models, algorithms and experiments.* PhD thesis, University of Texas at Austin, 2005.

[8] Vladimir Batagelj and Ulrik Brandes. Efficient Generation of Large Random Networks. *Physical Review E*, 71(3):36113, March 2005.

[9] Roberto Battiti and Alan Albert Bertossi. Greedy, Prohibition, and Reactive Heuristics for Graph Partitioning. *IEEE Transactions on Computers*, 48(4):361–385, April 1999.

[10] Roberto Battiti, Mauro Brunato, and Franco Mascia. *Reactive Search and Intelligent Optimization.* Operations research/Computer Science Interfaces. Springer Verlag, 2008.

[11] Roberto Battiti and Franco Mascia. Reactive and dynamic local search for max-clique: Engineering effective building blocks. *Computers & Operations Research*, 37(3):534–542, March 2010.

[12] Roberto Battiti and Marco Protasi. Reactive Local Search for the Maximum Clique Problem. Technical Report TR-95-052, ICSI, 1947 Center St.- Suite 600 - Berkeley, California, September 1995.

[13] Roberto Battiti and Marco Protasi. Reactive Local Search for the Maximum Clique Problem. *Algorithmica*, 29(4):610–637, 2001.

[14] Roberto Battiti and Giampietro Tecchiolli. The Reactive Tabu Search. *ORSA Journal on Computing*, 6(2):126–140, 1994.

[15] Ivano Bertini, Astrid Sigel, and Helmut Sigel, editors. *Handbook on Metalloproteins.* Marcel Dekker, New York, 1 edition, 2001.

[16] Mark Brockington and Joseph C. Culberson. Camouflaging independent sets in quasi-random graphs. In *Cliques, Coloring and Satisfiability: Second DIMACS Implementation Challenge*, volume 26 of *DIMACS Series*. American Mathematical Society, 1996.

[17] Mauro Brunato, Holger H. Hoos, and Roberto Battiti. On Effectively Finding Maximal Quasi-Cliques. In *Proceedings of the 2nd Learning and Intelligent Optimization Workshop*, Trento, 2009. Springer Verlag.

[18] Edmund Burke, Emma Hart, Graham Kendall, JimNewall, Peter Ross, and Sonia Schulenburg. Hyper-heuristics: an emerging direction in modern search technology. In Fred Glover, editor, *Handbook of Metaheuristics*, pages 457–474. Kluwer, 2003.

[19] Greg Burns, Raja Daoud, and James Vaigl. LAM: An Open Cluster Environment for MPI. In *Proceedings of Supercomputing Symposium*, pages 379–386, 1994.

[20] Sergiy Butenko and Wilbert E. Wilhelm. Clique-detection models in computational biochemistry and genomics. *European Journal of Operational Research*, 173:1–17, 2005.

[21] Konstantin Chakhlevitch and Peter I. Cowling. Hyper-heuristics: Recent Developments. In C Cotta, M Sevaux, and K Sörensen, editors, *Adaptive and Multilevel Metaheuristics*, volume 136 of *Studies in Computational Intelligence*, pages 3–29. Springer, 2008.

[22] Eric Chea and Dennis R Livesay. How accurate and statistically robust are catalytic site predictions based on closeness centrality? *BMC Bioinformatics*, 8:153+, May 2007.

[23] Marco Chiarandini and Franco Mascia. A hash function breaking symmetry in partitioning problems and its application to tabu search

for graph coloring. Technical Report 2010-025, IRIDIA, Université Libre de Bruxelles, Brussels, Belgium, December 2010.

[24] Elisa Cilia and Andrea Passerini. Automatic prediction of catalytic residues by modeling residue structural neighborhood. *BMC Bioinformatics*, 11(1):115, 2010.

[25] Stephen A. Cook. The complexity of theorem-proving procedures. In *STOC '71: Proceedings of the third annual ACM symposium on Theory of computing*, pages 151–158, New York, NY, USA, 1971. ACM.

[26] Peter Eades. A Heuristic for Graph Drawing. *Congressus Numerantium*, 42:149–160, 1984.

[27] Jessica C. Ebert and Russ B. Altman. Robust recognition of zinc binding sites in proteins. *Protein Sci*, 17(1):54–65, 2008.

[28] Thomas Finley and Thorsten Joachims. Supervised Clustering with Support Vector Machines. In *ICML*, 2005.

[29] Paolo Frasconi and Andrea Passerini. Predicting the Geometry of Metal Binding Sites from Protein Sequence. In *NIPS*, pages 465–472, 2008.

[30] Yaniv Frishman and Ayellet Tal. Online Dynamic Graph Drawing. *IEEE Transactions on Visualization and Computer Graphics*, 14(4):727–740, July 2008.

[31] Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness (Series of Books in the Mathematical Sciences).* W. H. Freeman & Co Ltd, January 1979.

[32] Andrea Grosso, Marco Locatelli, and Federico Della Croce. Combining swaps and node weights in an adaptive greedy approach for the maximum clique problem. *Journal of Heuristics*, 10:135–152, 2004.

[33] Andrea Grosso, Marco Locatelli, and Wayne J. Pullan. Randomness, plateau search, penalties, restart rules: simple ingredients leading to very efficient heuristics for the maximum clique problem. *Journal of Heuristics*, 2007.

[34] Johan Håstad. Clique is hard to approximate within $n^{1-\epsilon}$. In *Proceedings 37th Annual IEEE Symposium on Foundations of Computer Science*, pages 627–636. IEEE Computer Society, 1996.

[35] Aric Hagberg, Dan Schult, and Pieter Swart. No Title, 2004.

[36] Steven Halim and Roland H. C. Yap. Designing and Tuning SLS through Animation and Graphics: an Extended Walk-through. In *Proceedings of SLS 2007, Engineering Stochastic Local Search Algorithms. Designing, Implementing and Analyzing Effective Heuristics, International Workshop, Brussels, Belgium*, pages 16–30, 2007.

[37] Pierre Hansen, Nenad Mladenović, and Dragan Urošević. Variable neighborhood search for the maximum clique. *Discrete Applied Mathematics*, 145:117–125, 2004.

[38] Holger H. Hoos and Thomas Stützle. *Stochastic Local Search: Foundations and Applications.* Morgan Kaufmann Publishers, USA, 2004.

[39] Yongmei Ji, Xing Xu, and Gary D. Stormo. A graph theoretical approach to predict common RNA secondary structure motifs including pseudoknots in unaligned sequences. *Bioinformatics*, 20(10):1591–1602, 2004.

[40] Richard M. Karp. Reducibility Among Combinatorial Problems. In R E Miller and J W Thatcher, editors, *Complexity of Computer Computations*, pages 85–103. Plenum Press, 1972.

[41] Kengo Katayama, Akihiro Hamamoto, and Hiroyuki Narihisa. Solving the maximum clique problem by k-opt local search. In *Proceedings of the 2004 ACM Symposium on Applied computing*, pages 1021–1025, 2004.

[42] Mario Koppen and Kaori Yoshida. Visualization of Pareto-Sets in Evolutionary Multi-Objective Optimization. *Hybrid Intelligent Systems, 2007. HIS 2007. 7th International Conference on*, pages 156–161, 2007.

[43] Leonid A. Levin. Universal Sequential Search Problems. *Problems of Information Transmission*, 9(3), 1973.

[44] Elena Marchiori. Genetic, iterated and multistart local search for the maximum clique problem. In Stefano Cagnoni, Jens Gottlieb, Emma Hart, Martin Middendorf, and Günther Raidl, editors, *Applications of Evolutionary Computing*, volume 2279 of *Lecture Notes in Computer Science*, pages 112–121. Springer Berlin / Heidelberg, 2002.

[45] Franco Mascia and Mauro Brunato. Techniques and tools for local search landscape visualization and analysis. In *Proceedings of SLS 2009, Engineering Stochastic Local Search Algorithms. Designing, Implementing and Analyzing Effective Heuristics, International Workshop, Brussels, Belgium*, pages 92–104, 2009.

[46] Franco Mascia, Elisa Cilia, Mauro Brunato, and Andrea Passerini. Predicting Structural and Functional Sites in Proteins by Searching for Maximum-Weight Cliques. In *Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence (AAAI-10)*, pages 1274–1279. AAAI Press, July 2010.

[47] David W. Matula. *The Largest Clique Size in a Random Graph*. Dept. of Computer Science, Southern Methodist University, 1976.

[48] Panos M. Pardalos and Jue Xue. The maximum clique problem. *Journal of Global Optimization*, 4:301–328, 1994.

[49] Andrea Passerini, Marco Punta, Alessio Ceroni, Burkhard Rost, and Paolo Frasconi. Identifying cysteines and histidines in transition-metal-binding sites using support vector machines and neural networks. *Proteins*, 65(2):305–316, 2006.

[50] Pavel A. Pevzner and Sing-Hoi Sze. Combinatorial Approaches to Finding Subtle Signals in DNA Sequences. In *Proceedings of the Eighth International Conference on Intelligent Systems for Molecular Biology*, pages 269–278. AAAI Press, 2000.

[51] Hartmut Pohlheim. Visualization of evolutionary algorithms-set of standard techniques and multidimensional visualization. In *Proceedings of the Genetic and Evolutionary Computation Conference*, volume 1, pages 533–540. Morgan Kaufmann, 1999.

[52] Craig T. Porter, Gail J. Bartlett, and Janet M. Thornton. The Catalytic Site Atlas: a resource of catalytic sites and residues identified in enzymes using structural data. *Nucleic Acids Res*, 32(Database issue), January 2004.

[53] Wayne J. Pullan. Phased Local Search for the Maximum Clique Problem. *Journal of Combinatorial Optimization*, 12(3):303–323, November 2006.

[54] Wayne J. Pullan and Holger H. Hoos. Dynamic Local Search for the Maximum Clique Problem. *Journal of Artificial Intelligence Research*, 25:159–185, February 2006.

[55] Wayne J. Pullan, Franco Mascia, and Mauro Brunato. Cooperating local search for the maximum clique problem. *Journal of Heuristics*, 2010.

[56] Davood Rafiei and Stephen Curial. Effectively Visualizing Large Networks Through Sampling. In *WWW2005 Proceedings*, 2005.

[57] Nanjiang Shu, Tuping Zhou, and Sven Hovmoller. Prediction of zinc-binding sites in proteins from sequence. *Bioinformatics*, 24(6):775–782, 2008.

[58] Christine Solnon and Serge Fenet. A study of aco capabilities for solving the maximum clique problem. *Journal of Heuristics*, 12(3):155–180, 2006.

[59] Jeffrey M. Squyres and Andrew Lumsdaine. A Component Architecture for LAM/MPI. In *Proceedings, 10th European PVM/MPI Users' Group Meeting*, number 2840 in Lecture Notes in Computer Science, pages 379–387, Venice, Italy, September 2003. Springer-Verlag.

[60] Éric D. Taillard. Robust taboo search for the quadratic assignment problem. *Parallel computing*, 17(4-5):443–455, 1991.

[61] Wenxu Tong, Ying Wei, Leonel F. Murga, Mary Jo Ondrechen, and Ronald J. Williams. Partial Order Optimum Likelihood (POOL): Maximum Likelihood Prediction of Protein Active Site Residues Using 3D Structure and Sequence Properties. *PLoS Computational Biology*, 5(1):e1000266+, January 2009.

Part III

Appendix

Appendix A

A Comparison of Tabu Search Variations

RLS–LTM has been described in Chapter 4 as a more effective implementation of of the original RLS for MC [13]. Faster restarts and greater diversification allowed to improve over the original RLS for MC, but also caused the explosion of the value of T on some hard instances. In Chapter 4 it is conjectured that with a high value of T it is unlikely that all nodes belonging to the maximum clique are not prohibited and can be added to the current configuration, therefore an upper bound $\mathsf{MAX\_T} = 0.5(|\mathsf{Best}| + 1)$ has been introduced.

In this chapter we present an in-depth analysis of the dynamics of RLS and RLS–LTM. We study how the reactive mechanism impacts on the overall performances, and its effectiveness in tuning the tabu tenure for the instance at hand and the local characteristics of the search landscape.

A.1 Peeking Under the Hood of RLS–LTM

Reactive Tabu Search is a meta-heuristic that adapts the tabu tenure T of the underlying Tabu Search. It automatically tunes the parameter T for the instance at hand, and more importantly it adapts it throughout the

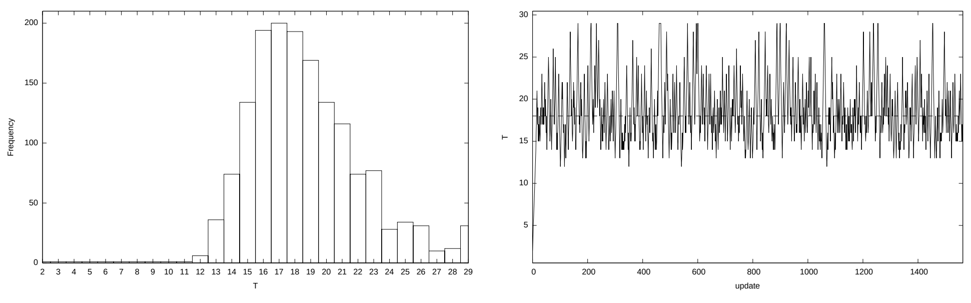


Figure A.1: Adaptation of tabu tenure T for the instance C500.9 (RLS–LTM). The graph on the left shows how many times the parameter T has been set to a particular value. The graph on the right shows the evolution of the parameter in time. The dashed line shows the median value.

run by reacting on the local characteristics of the search space.

The aim of this study is at understanding how effective is the adaptation of the tabu tenure in the case of RLS–LTM for the MC problem. More specifically we want to understand if RLS–LTM is rapidly converging to the best parameter T for the instance at hand, or if it is adapting it to different values depending on the local characteristic of the search space.

In order to understand how the tabu tenure is adapted, we run the algorithm on the DIMACS benchmark instances and trace the history of the parameter T throughout 1,000,000 iterations. Figure A.1, A.2, A.3, and A.4 are representative of the four different pictures we got across the benchmark set. In Figure A.1 the value converges immediately and stays around the average with small oscillations. Also in Figure A.2 the parameter converges in few iterations but the distribution of T is saturated on the upper-bound MAX_T. The opposite happens in Figure A.3 where the tenure is updated rarely, and for most of the iterations the parameter T remains on the minimum value allowed. There are very few spikes corresponding to repetitions encountered during the search. On some instances, like in the case of C4000.5 in Figure A.4, the picture is less clear.

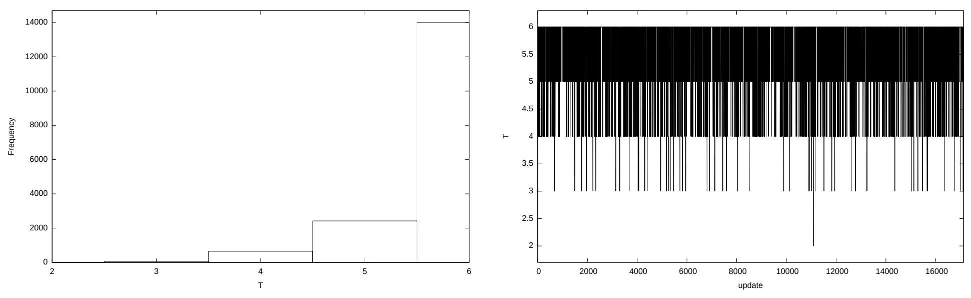


Figure A.2: Adaptation of tabu tenure T for the instance brock200_4 (RLS–LTM). The graph on the left shows how many times the parameter T has been set to a particular value. The graph on the right shows the evolution of the parameter in time. The dashed line shows the median value.

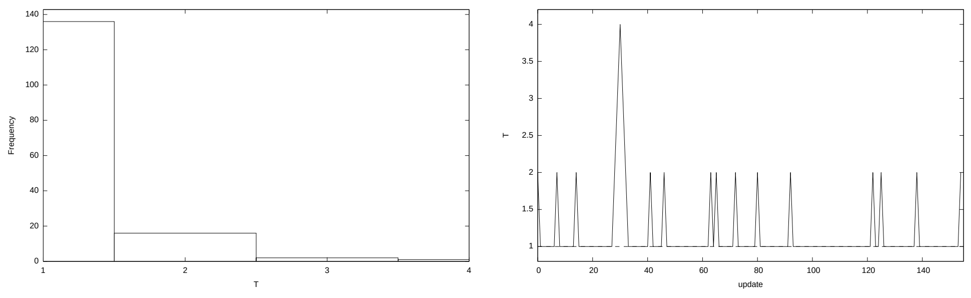


Figure A.3: Adaptation of tabu tenure T for the instance MANN_a45 (RLS–LTM). The graph on the left shows how many times the parameter T has been set to a particular value. The graph on the right shows the evolution of the parameter in time. The dashed line shows the median value.

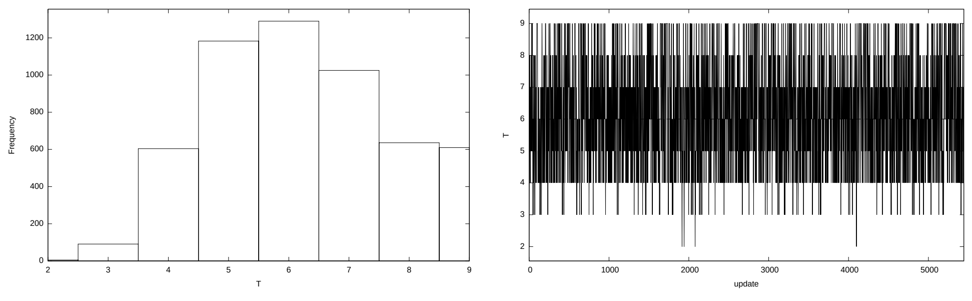


Figure A.4: Adaptation of tabu tenure T for the instance C4000.5 (RLS–LTM). The graph on the left shows how many times the parameter T has been set to a particular value. The graph on the right shows the evolution of the parameter in time. The dashed line shows the median value.

The overall picture emerging is the following: either there is a strong threshold effect, or the parameter T quickly converges to an average value with little oscillations around it during the search.

Looking at the instances with an evident threshold effect, it is clear that the adaptation of the tabu tenure is not optimal at least in those cases. To better understand how the reactive mechanism impacts on the performances we try to update at each iteration the tenure T to a random value in the interval [MIN_T, MAX_T]. Figure A.5 shows the performance of a reactive and a random tenure update on the subset of the DIMACS benchmark set where both find the maximum clique on all runs. The maximum computational budget allocated is 100,000,000 iterations. Each dot in the graph represents the median number of iterations over 100 successful runs on a single instance. The performances are highly correlated: the Spearman's rank order test rejects the hypothesis of no significant (monotone) relationship between the samples with p-value $1.67 \cdot 10^{-20}$. The empty dots represent instances for which a Mann-Whitney U-test could not reject the null hypothesis of the algorithms having an identical performance at significance level $\alpha = 0.05$. The white squares on the axes show instances

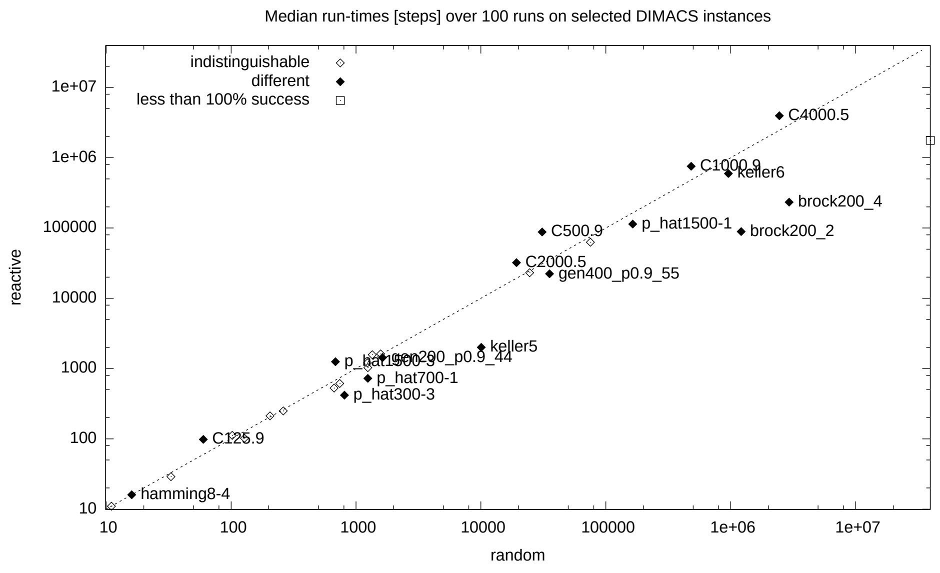


Figure A.5: Median number of steps to converge to the optimal solution when setting the tenure randomly or reactively. The empty dots represent the instances for which a Mann-Whitney U-test could not reject the null hypothesis of the algorithms having an identical performance at significance level $\alpha = 0.05$. The white squares on the axes show instances for which the one of the algorithm was not able to find the maximum clique every run.

for which the one of the algorithm was not able to find the maximum clique every run. For the other cases the name of the instances is reported. The only notable differences are on the brock200 instances. A Wilcoxon matched pairs signed rank test on the medians could not reject the null hypothesis of the algorithms having an identical performance at significance level $\alpha = 0.05$. The same conclusions can be drawn also looking at the CPU seconds in Figure A.6. The detailed results are shown in table A.1.

The quite surprising results can be explained by the small interval $[\mathsf{MIN\_T}, \mathsf{MAX\_T}]$. Such a small interval makes the reactive mechanism statistically indistinguishable from a Tabu Search selecting a tabu value randomly around $\mathsf{MIN\_T}\frac{\mathsf{MAX\_T}-\mathsf{MIN\_T}}{2}$. The only cases in which the performance are worse are the brock200 instances in which a stronger diversification would be necessary, and the random distribution of the values of T

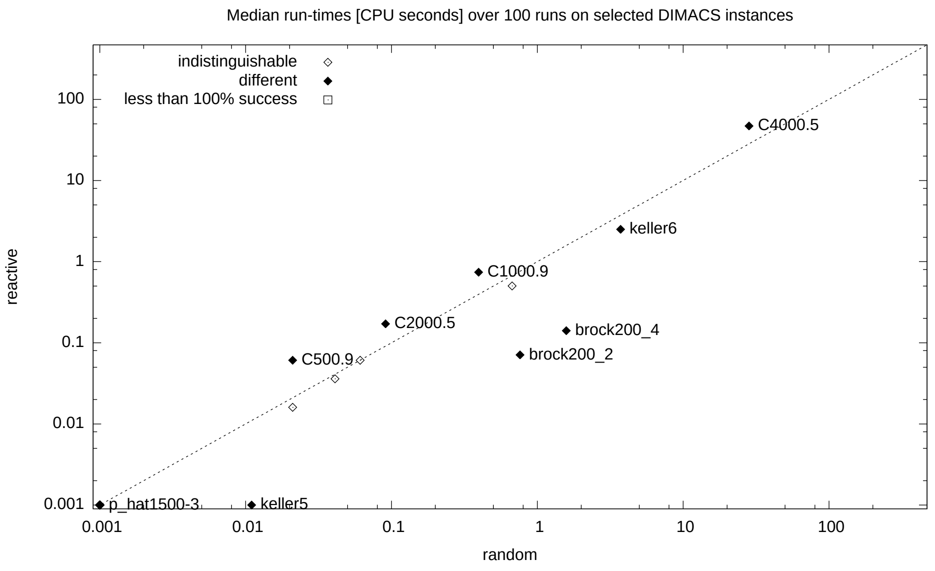


Figure A.6: Median number of CPU seconds to converge to the optimal solution when setting the tenure randomly or reactively. The empty dots represent the instances for which a Mann-Whitney U-test could not reject the null hypothesis of the algorithms having an identical performance at significance level $\alpha = 0.05$. The white squares on the axes show instances for which the one of the algorithm was not able to find the maximum clique every run.

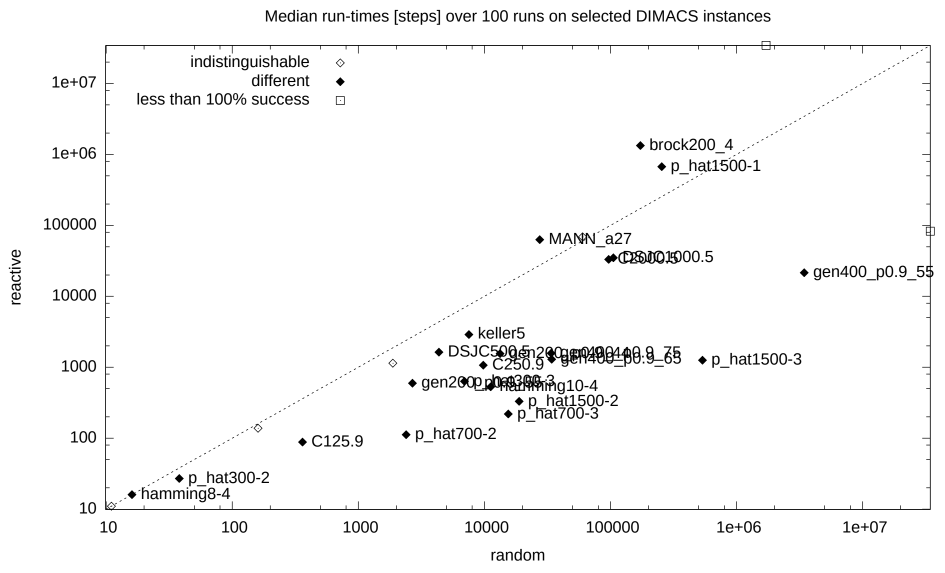


Figure A.7: Median number of steps to converge to the optimal solution when setting the tenure randomly or reactively and MAX_T $= 2(|\mathsf{Best}| + 1)$. The empty dots represent the instances for which a Mann-Whitney U-test could not reject the null hypothesis of the algorithms having an identical performance at significance level $\alpha = 0.05$. The white squares on the axes show instances for which the one of the algorithm was not able to find the maximum clique every run.

is centred far from the threshold MAX_T.

In order to understand the impact of MAX_T on the performances of the two heuristics we increase its value from $0.5(|\mathsf{Best}|+1)$ to $2(|\mathsf{Best}|+1)$, and run the same experiments again.

Figure A.7, and A.8 show the performance of both implementations on the subset of the DIMACS benchmark set where both find the maximum clique on all runs. For example, the plots do not show the median values for the instance C4000.5, since the algorithm which sets the tenure randomly finds the optimum solution only in the 93% of the runs, and the algorithm setting the tenure reactively performs even worse, being able to find the optimum only in 30% of the runs. The detailed results are shown in Table A.2.

With a less stringent upper bound on the tenure value, the perfor-

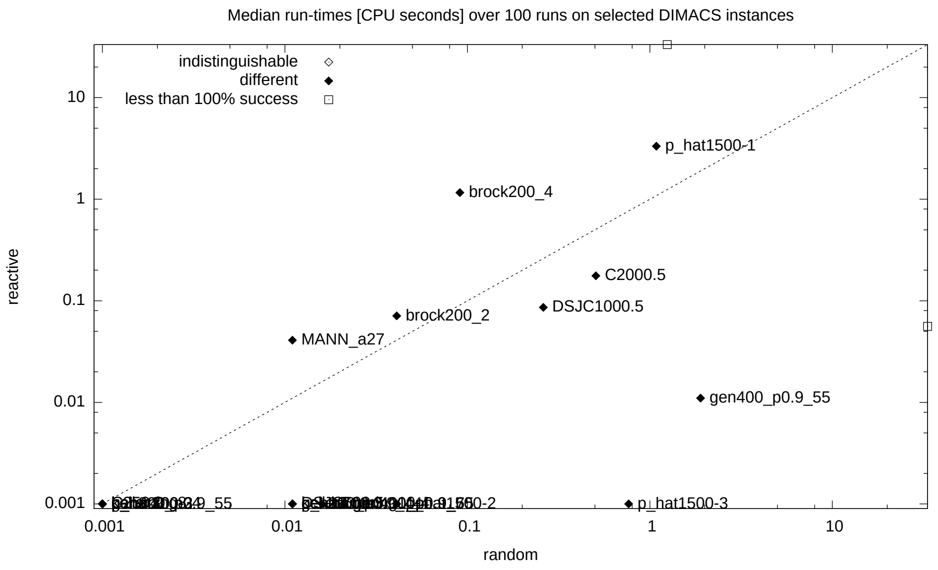


Figure A.8: Median number of CPU seconds to converge to the optimal solution when setting the tenure randomly or reactively and $\mathsf{MAX\_T} = 2(|\mathsf{Best}| + 1)$. The empty dots represent the instances for which a Mann-Whitney U-test could not reject the null hypothesis of the algorithms having an identical performance at significance level $\alpha = 0.05$. The white squares on the axes show instances for which the one of the algorithm was not able to find the maximum clique every run.

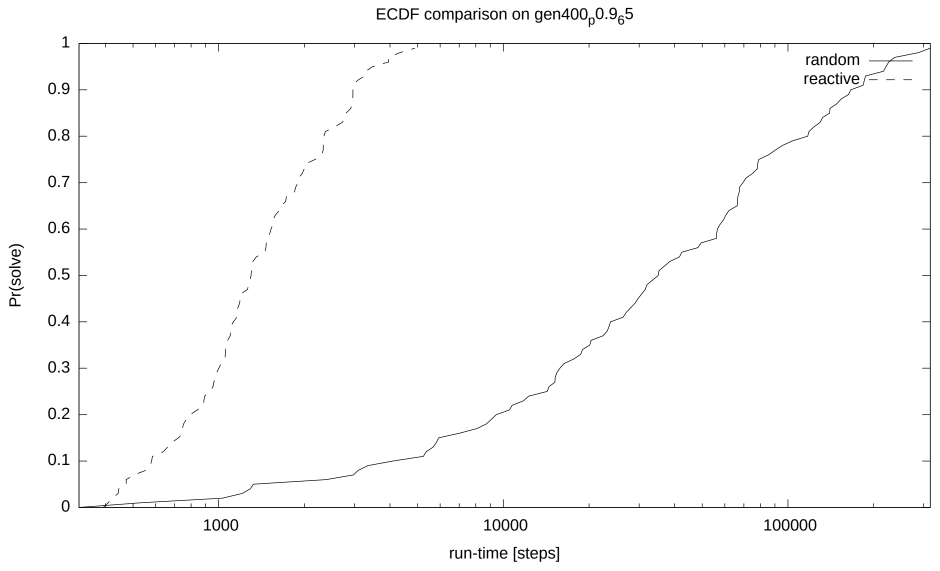


Figure A.9: Empirical QRTDs on the gen400_p0.9_65 instance when MAX_T $= 2(|\mathsf{Best}| + 1)$.

mances of both algorithm are deteriorated, meaning that both heuristics are strongly sensible to the meta-parameter MAX_T. This is especially true when the tenure value is adjusted randomly, since the distribution of the tenure values is spread on a larger interval and centred on a value which is 4 times bigger in the case of of MAX_T $= 0.5(|\mathsf{Best}| + 1)$. The consequent search stagnation is evident in Figure A.9.

On the contrary, in the instances where there are many repeated configurations, the reactive algorithm saturates the parameter T to a too high upper bound MAX_T, while the lower average value for the random setting performs better as depicted in Figure A.10.

The performances are strongly influenced by the selection of the right value for T, this is evident looking at the position of the brock200 instances in Figure A.6 and A.8 where in the first case the reactive algorithm was limited by a too stringent MAX_T and the random selection performed even worse with a lower average value. In the second case, while the reactive algorithm focusses on too high values the random one finds a better average

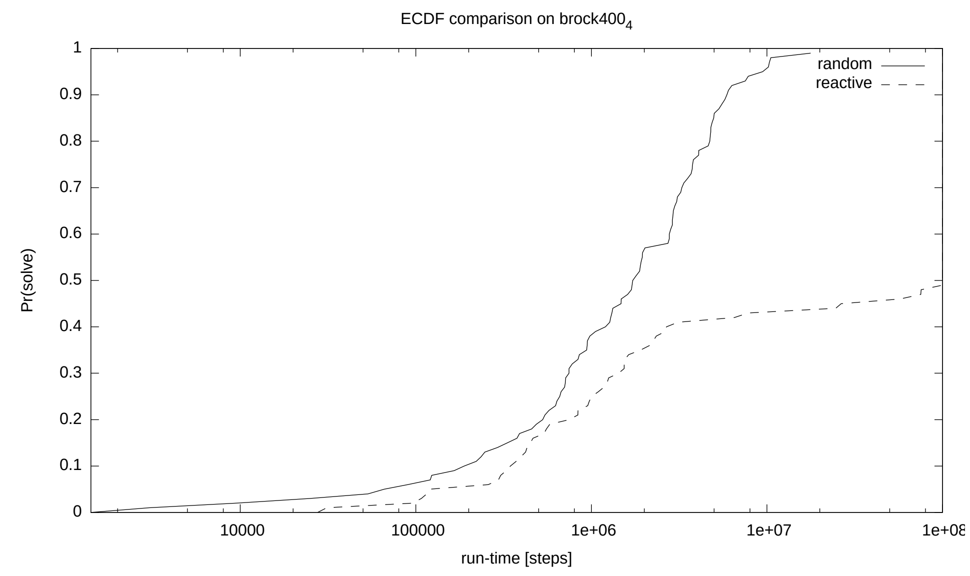


Figure A.10: Empirical QRTDs on the brock400_4 instance when $\mathsf{MAX\_T} = 2(|\mathsf{Best}| + 1)$.

T.

Table A.5 shows a pairwise comparison between reactive and random settings of the tenure with the two different upper-bounds seen so far.

This means that the reactive mechanism in RLS–LTM is not able to find a good value for T without the correct $\mathsf{MAX\_T}$ meta-parameter, because it is somehow tricked by the high number of repeated configurations encountered. But how does the 'wrong' value of T impact on the search dynamics? A possible answer to this question is that the impact is given by the reduced number of choices that the algorithm can make due to the too many prohibited nodes. The reduction in the number of available choices, if confirmed, should have a stronger impact on the performances, than the original assumption that lead to the introduction of $\mathsf{MAX\_T}$, i.e., that for high values of T it was improbable to have all nodes belonging to the optimal solution to be non-prohibited. Hints that this intuition could be correct come from at least the case of the instance C4000.5, in which the increase of $\mathsf{MAX\_T}$ rendered the reactive algorithm incapable of finding the

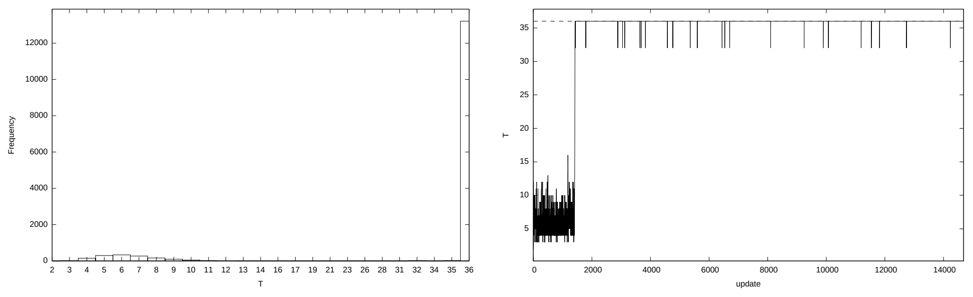


Figure A.11: Adaptation of tabu tenure T for the instance C4000.5 when $\mathsf{MAX\_T} = 2(|\mathsf{Best}|+1)$. The graph on the left shows how many times the parameter T has been set to a particular value. The graph on the right shows the evolution of the parameter in time. The dashed line shows the median value.

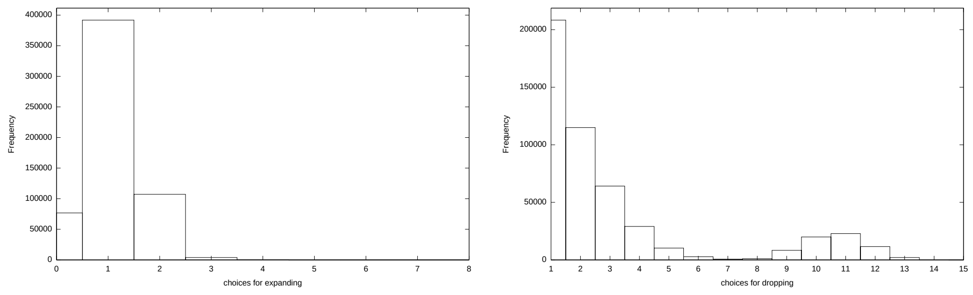


Figure A.12: Number of non prohibited nodes during add or drops for RLS–LTM when $\mathsf{MAX\_T} = 0.5(|\mathsf{Best}| + 1)$.

maximum clique. The explosion the tabu tenure is visible in Figure A.11, especially when compared to Figure A.4. Figure A.12 and A.13 show the choices that are available for expanding or dropping a node, when $\mathsf{MAX\_T}$ is equal to $0.5(|\mathsf{Best}|+1)$ or $2(|\mathsf{Best}|+1)$ respectively. The pictures show that although there is no much difference in the intensification, the increased T value has a strong negative impact on the diversification.

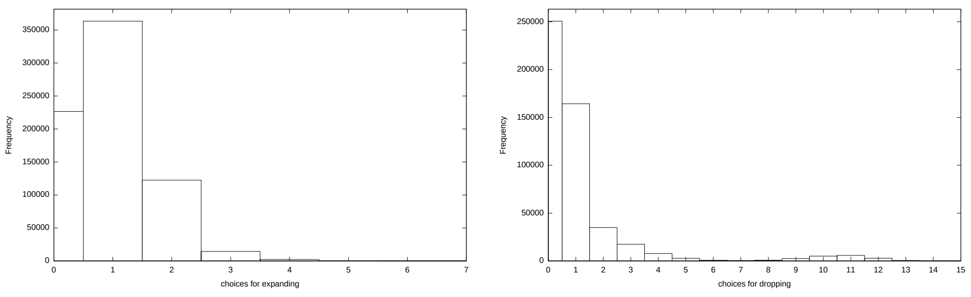


Figure A.13: Number of non prohibited nodes during add or drops for RLS–LTM when $\mathsf{MAX\_T} = 2(|\mathsf{Best}| + 1)$.

A.2 Long vs. Short Term Memory

The results presented in Chapter 4 show the improved performance of RLS–LTM over RLS both in terms of steps per second and number of steps to converge to the optimum; but it has never been analysed how the algorithmic and the implementation changes contribute to the performance.

We are particularly interested in the algorithmic changes here, since they also introduce undesired effects on the explosion of the tenure T, and the consequent need for the introduction of a new upper bound.

We run RLS–LTM without the algorithmic changes in order to measure the performances of a possible efficient implementation of the original RLS. From Figure A.15 and A.16 it emerges that other things being equal the algorithmic changes introduced in RLS–LTM improve the overall performances. The improvement measured on the DIMACS benchmark set and detailed in Table A.3 is due almost exclusively to the faster restarts. In fact on most instances there is no difference in the median number of steps to reach the optimum, but there is up to an order of magnitude in the CPU seconds. Figure A.14 shows a comparison of the the empirical run time distributions on the keller6 instance where RLS–LTM performs in median half of the iterations than RLS in less than 5% of the CPU seconds.

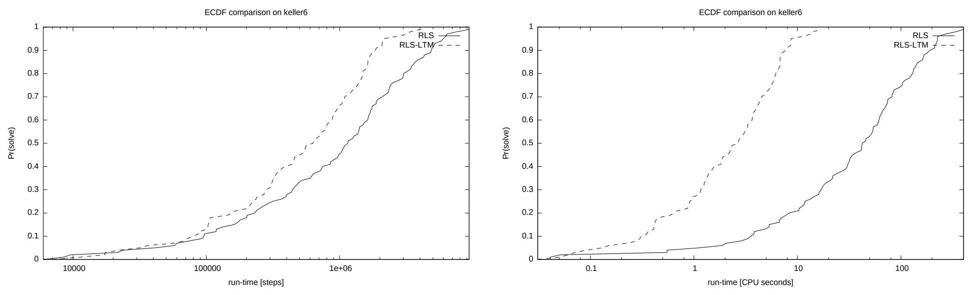


Figure A.14: Empirical QRTDs on the keller6 instance.

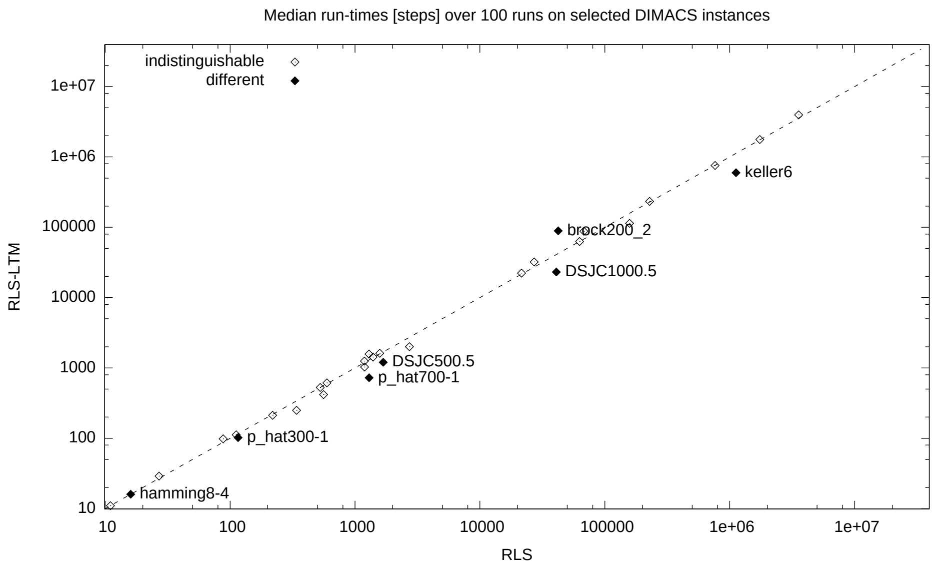


Figure A.15: Median number of steps to converge to the optimal solution of efficient implementations of RLS and RLS–LTM. The empty dots represent the instances for which a Mann-Whitney U-test could not reject the null hypothesis of the algorithms having an identical performance at significance level $\alpha = 0.05$.

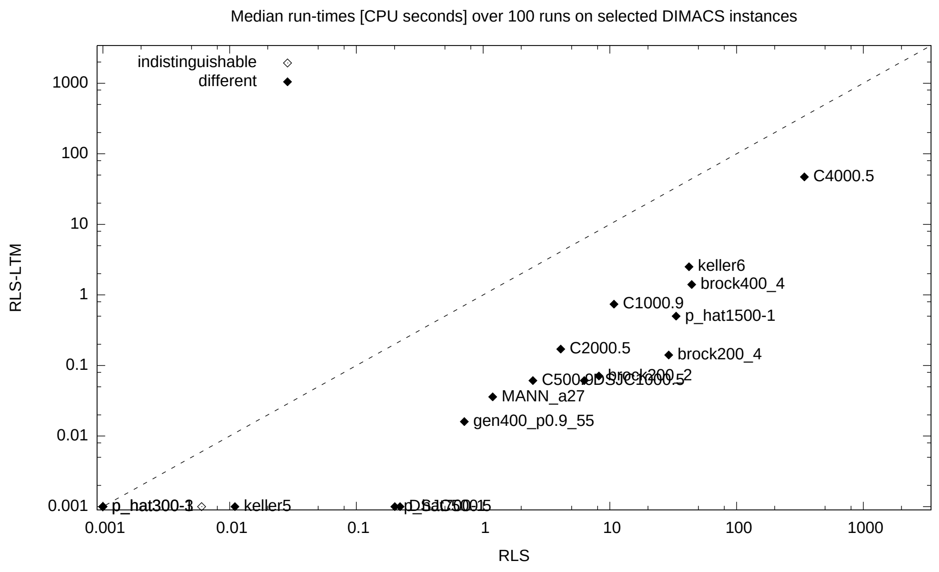


Figure A.16: Median number of CPU seconds to converge to the optimal solution of efficient implementations of RLS and RLS–LTM. The empty dots represent the instances for which a Mann-Whitney U-test could not reject the null hypothesis of the algorithms having an identical performance at significance level $\alpha = 0.05$.

Figure A.17, A.18, A.19, and A.20 show the adaptation of the tabu tenure of RLS in the four cases discussed at the beginning of this Chapter, namely C500.9, brock200_4, C4000.5, and MANN_a45. Figure A.19 is almost identical to Figure A.3 depicting the adaptation of the tenure of RLS–LTM. The small number of repetitions in the search history keeps the tabu tenure on smaller values. Also for the instance C4000.5 (Figure A.20) RLS keeps the tenure T around small values, but in the case of the hardest brock200_4 (Figure A.18) the explosion of T before the restarts is quite evident. In the instance C500 depicted in Figure A.17 the tenure does not explode, and the high number of restarts can be seen from frequency of the smaller values of T.

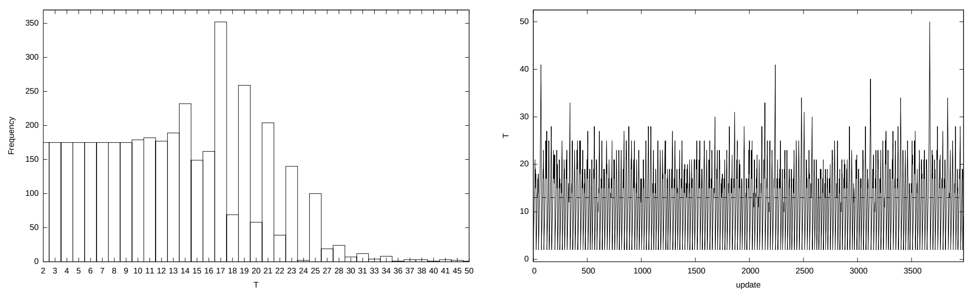


Figure A.17: Adaptation of tabu tenure T for the instance C500.9. The graph on the left shows how many times the parameter T has been set to a particular value. The graph on the right shows the evolution of the parameter in time. The dashed line shows the median value.

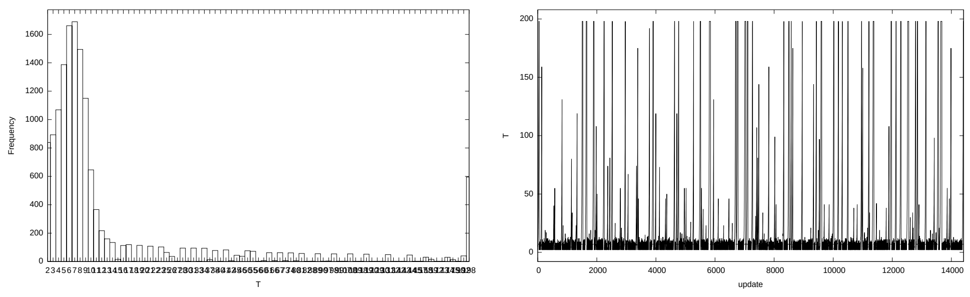


Figure A.18: Adaptation of tabu tenure T for the instance brock200_4. The graph on the left shows how many times the parameter T has been set to a particular value. The graph on the right shows the evolution of the parameter in time. The dashed line shows the median value.

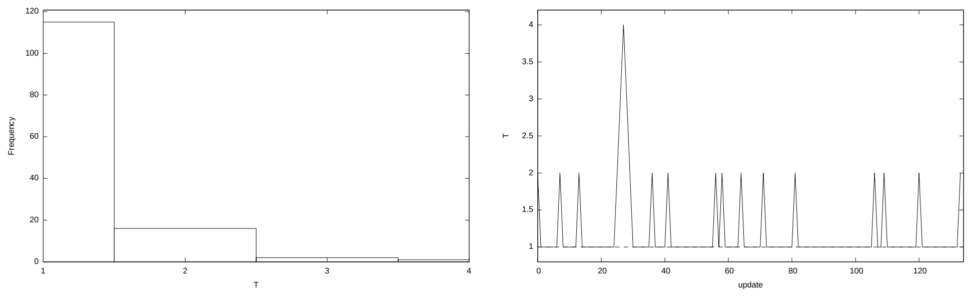


Figure A.19: Adaptation of tabu tenure T for the instance MANN_a45. The graph on the left shows how many times the parameter T has been set to a particular value. The graph on the right shows the evolution of the parameter in time. The dashed line shows the median value.

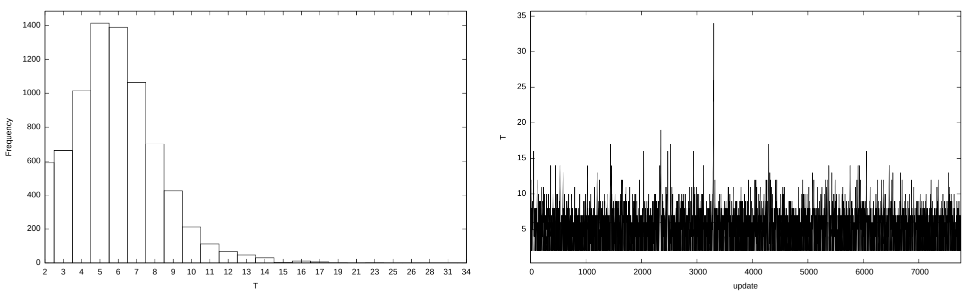


Figure A.20: Adaptation of tabu tenure T for the instance C4000.5. The graph on the left shows how many times the parameter T has been set to a particular value. The graph on the right shows the evolution of the parameter in time. The dashed line shows the median value.

A.2.1 A Good Tabu Tenure

Figure A.43, A.47, A.46, and A.52 show the mean number of iterations to find the optimum clique over 10 runs for the instances C500.9, brock200_4, MANN_a45, and C4000.5 respectively. The algorithm used to measure the number of iterations is a Tabu search with the same operations of RLS (expand, drop, and restart) and a fixed parameter for the tabu tenure T, and for the restart frequency R. The iteration budget in the 10 runs is fixed to 10,000,000. When the algorithm reaches the maximum amount of iterations, it means that it could not find the optimum solution. In most cases there is a setting for the value of T for which the restart frequency is not critical. There are few exceptions, e.g. very hard instances like MANN_a45 in Figure A.46 and MANN_a27 in Figure A.45 in which the opposite is true, i.e., the performances strongly depend on the restart frequency and much less from the value of the tabu tenure T. The restarts make the whole algorithm more robust: except for some extreme values, restarts do not worsen the performances in the instances for which a right value for the tenure is essential, and can be decisive in other instances.

Looking at the histograms in Figure A.17, A.18, A.19, and A.20 and comparing the mode of the tenure T with the plots in Figure A.43, A.47, A.46, and A.52 it's clear that RLS is able to spot the most appropriate value of T for the particular instance.

The Bias in the Restarts

RLS is able to find a good value of T in most of the cases, but is it because of the effectiveness of the reaction or is it because the frequent restarts mitigate the tenure value?

In [13] restarts were introduced to deal with disconnected part of the search space, but actually without restarts that periodically reset the value of T, RLS could sometimes miss the right value of T for the instance at hand. This is quite noticeable in brock200_4 (Figure A.18) with the peak on 198 in the histogram (maximum value reachable by T in the experiment $|V| - 2$).

To further investigate this point we run a version of RLS in which there is no upper bound MAX_T and no restarts. Figure A.22 shows a run on brock200_4 where the explosion of the tenure T is confirmed. The issue is noticeable not only in *small* and hard instances but also on small very *simple* instances where this effect would go unnoticed because of the very few steps required to find the optimum. For example in Figure A.21 the graph C125.9 has 125 nodes and the maximum clique has size 34, RLS is able to find the max clique in few hundred steps and the explosion of the parameter would be unnoticed. The same can be said for other simple instances, e.g., hamming8-4 in Figure A.23. Among the same family of instances it happens just on the easiest ones. For hard instances like C4000.5 or keller5 the average value of T determined by RLS (Figure A.24 and A.25) lies among the best ones possible for the instance (Figure A.52 and A.51 respectively).

C4000.5 is the most surprising results, in Figure A.11 RLS–LTM clearly shows the explosion of the tenure and the consequent performance degra-

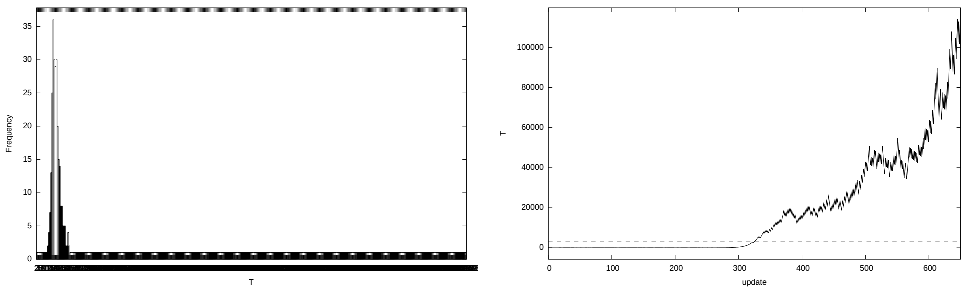


Figure A.21: Adaptation of tabu tenure T for the instance C125.9 with no restarts and no MAX_T. The graph on the left shows how many times the parameter T has been set to a particular value. The graph on the right shows the evolution of the parameter in time. The dashed line shows the median value.

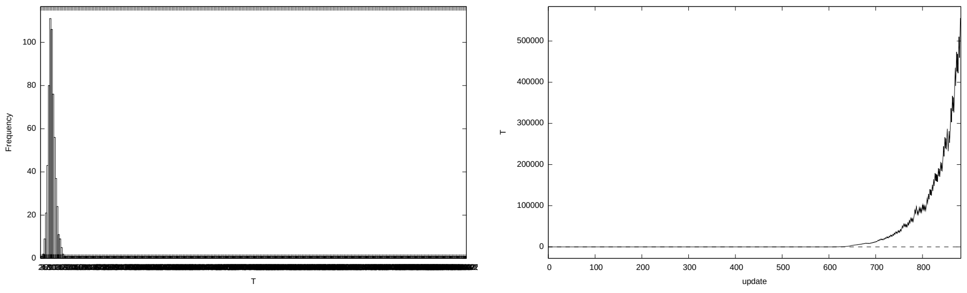


Figure A.22: Adaptation of tabu tenure T for the instance brock200_4 with no restarts and no MAX_T. The graph on the left shows how many times the parameter T has been set to a particular value. The graph on the right shows the evolution of the parameter in time. The dashed line shows the median value.

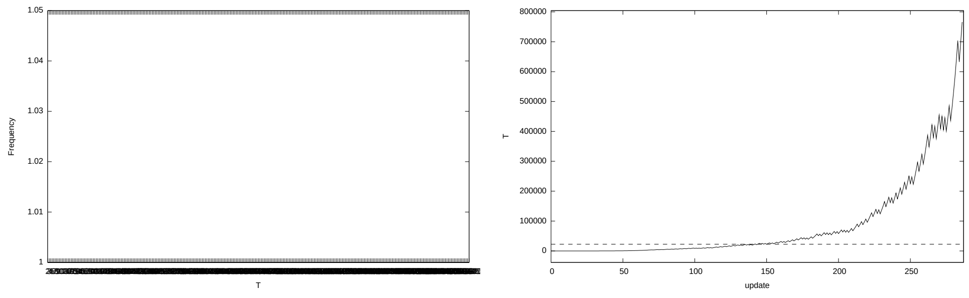


Figure A.23: Adaptation of tabu tenure T for the instance hamming8-4 with no restarts and no MAX_T. The graph on the left shows how many times the parameter T has been set to a particular value. The graph on the right shows the evolution of the parameter in time. The dashed line shows the median value.

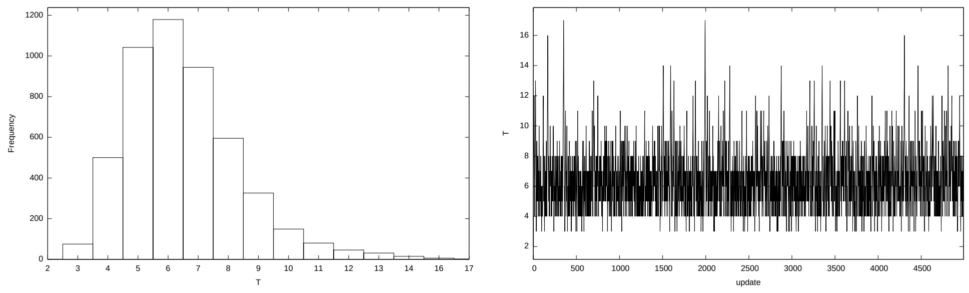


Figure A.24: Adaptation of tabu tenure T for the instance C4000.5 with no restarts and no MAX_T. The graph on the left shows how many times the parameter T has been set to a particular value. The graph on the right shows the evolution of the parameter in time. The dashed line shows the median value.

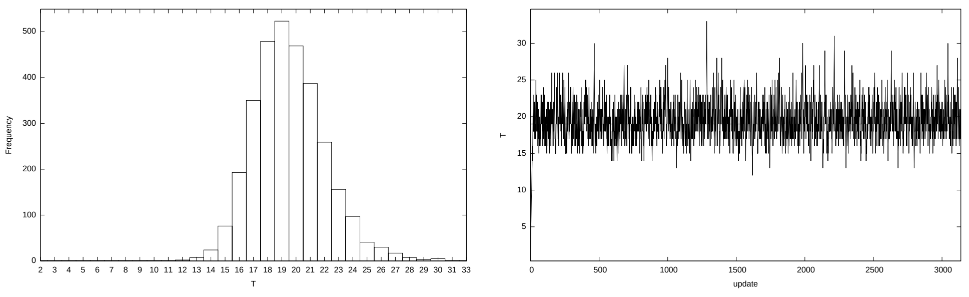


Figure A.25: Adaptation of tabu tenure T for the instance keller5 with no restarts and no MAX_T. The graph on the left shows how many times the parameter T has been set to a particular value. The graph on the right shows the evolution of the parameter in time. The dashed line shows the median value.

dation when MAX_T is not small enough. RLS has not this issue as shown in Figure A.20. One can conjecture that the difference is due to the only difference between RLS and RLS–LTM, i.e., the frequent restarts in RLS also reset the tenure T and this prevents the explosion. Oddly, when no upper-bound is set, and without the restarts, the parameter stays centred around good values of T for this instance, see Figure A.24. This effect does not depend from the upper bound but from the restarts. In fact, adding the restarts leads to the tenure explosion in both cases but in the case of RLS–LTM there is nothing that keeps the value of T to small values like for RLS. The root of this issue lies in the bias in the selection of the node seeding the current configuration. In fact, RLS empties the current configuration during the restarts and seeds it with a node that has never been included in a configuration and with the highest degree. Ties are broken randomly. The problem does not occur if one selects the node seeding the current clique randomly regardless of their degree and regardless of the fact that they have already been part of a solution.

The bias in the restart that is noticeable on certain instances in RLS–LTM could also affect the diversification and therefore the performances of

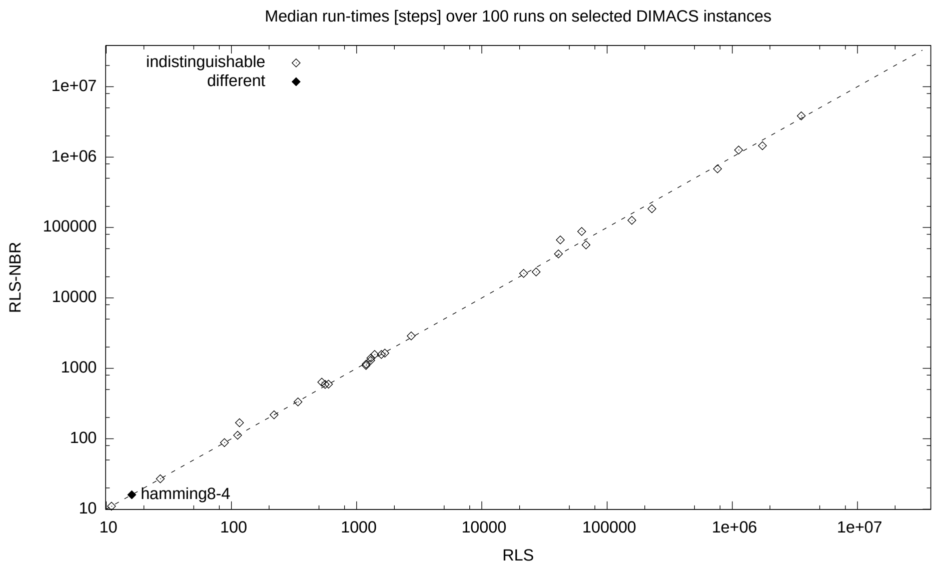


Figure A.26: Median number of steps to converge to the optimal solution of efficient implementations of RLS and RLS-NBR. The empty dots represent the instances for which a Mann-Whitney U-test could not reject the null hypothesis of the algorithms having an identical performance at significance level $\alpha = 0.05$.

RLS. In order to check if this is the case, we compare the median steps to reach the optimum solution for RLS and RLS-NBR. Figure A.26 compares the two restart implementation showing that the small differences are not statistically significant.

Fixed Tenure T

If we compare RLS with an implementation having the same restart frequency and the best tabu tenure T for the given instance the median steps to converge to the optimal solution is slightly smaller on most instances (see Figure A.27). The difference in the median CPU seconds in Figure A.28 is mostly due to the the cost of resetting the hash table during the restarts.

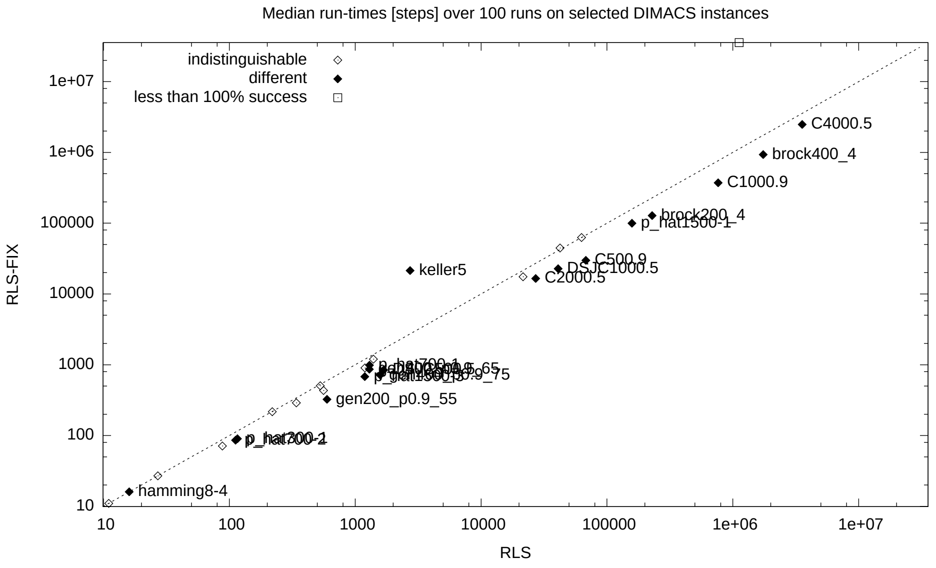


Figure A.27: Median number of steps to converge to the optimal solution of efficient implementations of RLS and RLS with best fixed T for the instance. The empty dots represent the instances for which a Mann-Whitney U-test could not reject the null hypothesis of the algorithms having an identical performance at significance level $\alpha = 0.05$. The white squares on the axes show instances for which the one of the algorithm was not able to find the maximum clique every run.

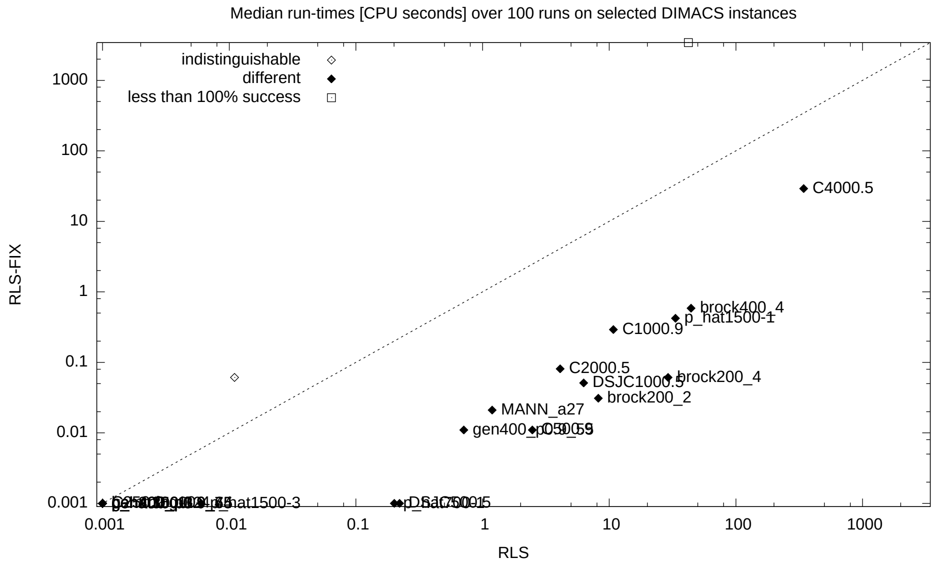


Figure A.28: Median number of CPU seconds to converge to the optimal solution of efficient implementations of RLS and RLS with best fixed T for the instance. The empty dots represent the instances for which a Mann-Whitney U-test could not reject the null hypothesis of the algorithms having an identical performance at significance level $\alpha = 0.05$. The white squares on the axes show instances for which the one of the algorithm was not able to find the maximum clique every run.

A.3 A New Implementation

We have seen that RLS–LTM can be tricked by too much memory, and without an upper-bound MAX_T the tabu tenure T explodes preventing the algorithm to find the optimum solution. RLS clears the search history and resets the tabu tenure to MIN_T at every restart, therefore it does not show the same parameter explosion as RLS–LTM. Clearing the hash table at every restarts is responsible for worse performances of RLS when compared to RLS–LTM. Even if the median number of steps for the two implementations to find the optimum solution are very similar, the difference in CPU seconds can be of one order of magnitude. Moreover, the bias in the restarts towards highly connected nodes that have never been part of a solution is accountable for the explosion of the tabu tenure in RLS–LTM on some instances. Also in this case, it does not happen in RLS where the restart reset the tabu tenure T.

Building on the analysis presented in the previous sections we build a new implementation RLS–fast in which the restarts are performed seeding the new solution with a completely random node, there is no artificial bound MAX_T, and during the restarts the tabu tenure is reset and the search history cleared in an efficient manner.

The hash table in RLS–LTM starts with 2^{24} elements and if necessary it doubles them every time the fill factor of the table is greater than 0.6. Conflicts are resolved with chaining, and the elements in the chains are are picked up from a pool of pre-allocated memory to avoid expensive system calls for memory allocations during the search.

In RLS–fast the hash table size is fixed to two million elements and does not grow during the search. There is no chaining for resolving conflicts, and locations holding elements older than the last restart are considered empty. Clearing the hash table is as fast as storing the time of the last

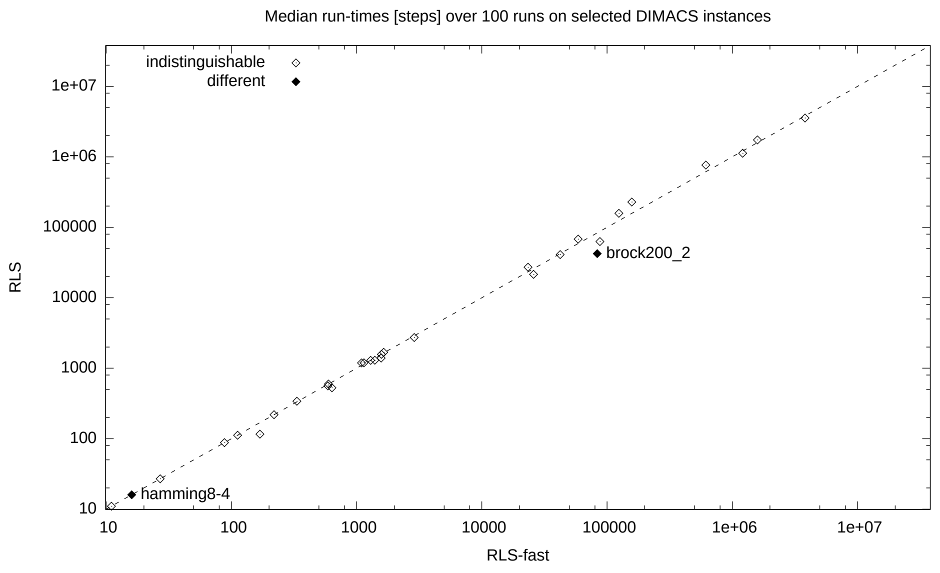


Figure A.29: Median number of steps to converge to the optimal solution of efficient implementations of RLS and RLS–fast. The empty dots represent the instances for which a Mann-Whitney U-test could not reject the null hypothesis of the algorithms having an identical performance at significance level $\alpha = 0.05$.

restart. Since there is no chaining when the hash table is full a look up operation could be as expensive as $O(n)$ where n is the size of the table. Therefore lookup in the hash table will stop as soon as one expired element is found.

The lookup operations can have false positive since only the hash of the solution are stored in the table, and also false negative because if a solution is stored far from its location because of conflicts there is a probability that after some iterations some location in between will expire.

Figure A.29 shows how the median number of iteration for reaching the optimum solution for RLS and RLS–fast is for most instances statistically indistinguishable. The faster restarts allow for improving the performances reaching those of RLS–LTM without the undesirable effects of keeping the search history across restarts (see Figure A.30). Figure A.31 and A.32 show the same comparison between RLS–fast and RLS–LTM.

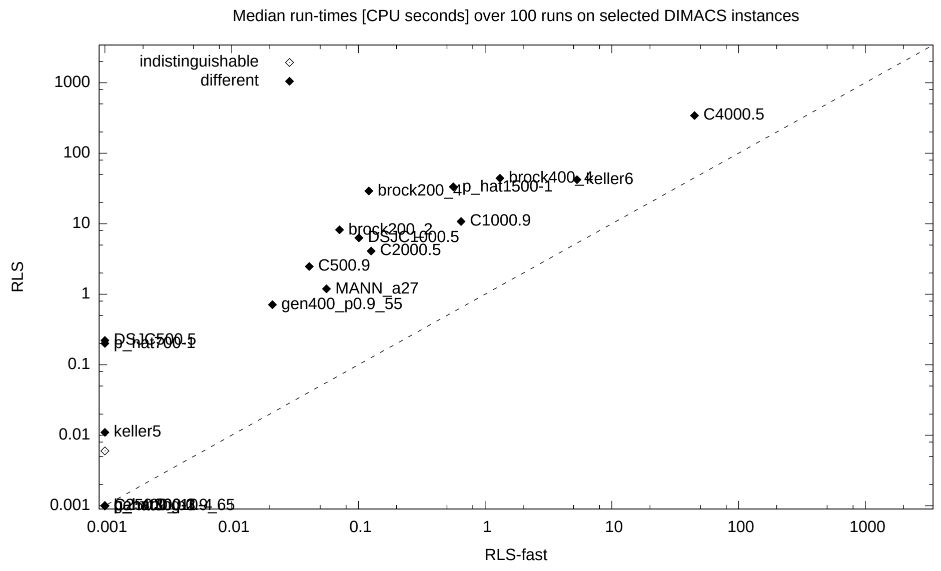


Figure A.30: Median number of CPU seconds to converge to the optimal solution of efficient implementations of RLS and RLS–fast. The empty dots represent the instances for which a Mann-Whitney U-test could not reject the null hypothesis of the algorithms having an identical performance at significance level $\alpha = 0.05$.

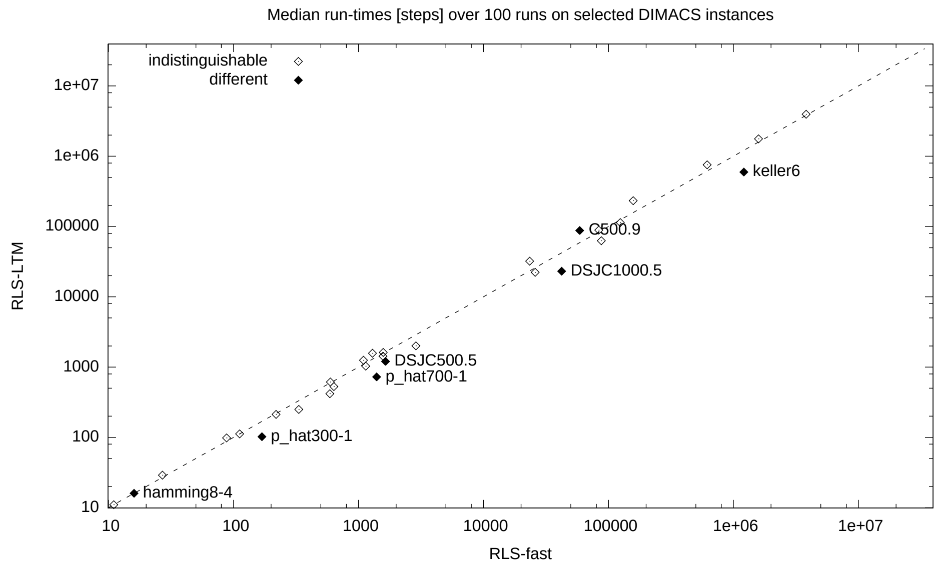


Figure A.31: Median number of steps to converge to the optimal solution of RLS–LTM and RLS–fast. The empty dots represent the instances for which a Mann-Whitney U-test could not reject the null hypothesis of the algorithms having an identical performance at significance level $\alpha = 0.05$.

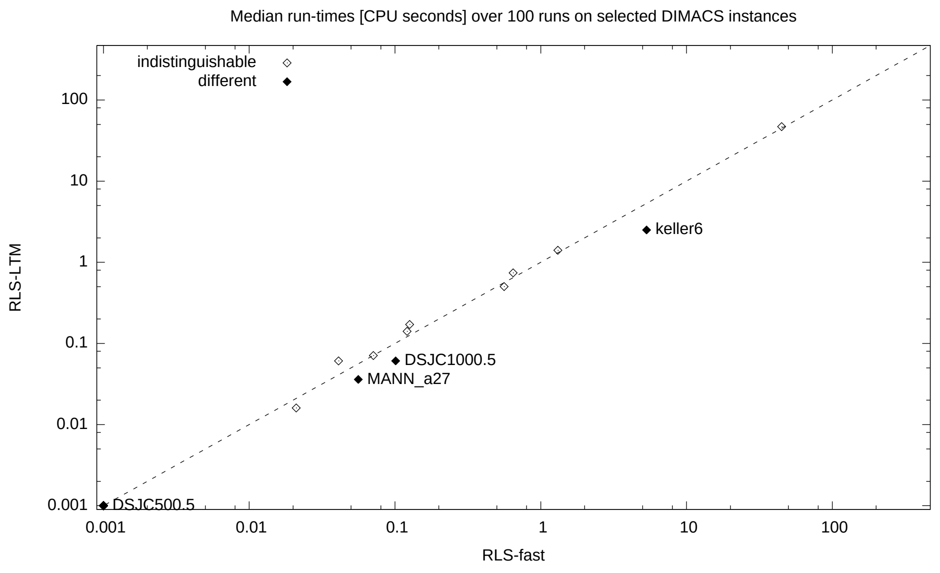


Figure A.32: Median number of CPU seconds to converge to the optimal solution of RLS–LTM and RLS–fast. The empty dots represent the instances for which a Mann-Whitney U-test could not reject the null hypothesis of the algorithms having an identical performance at significance level $\alpha = 0.05$.

Figure A.33 and A.34 compare the performances of RLS–fast with a version having the same restart frequency as RLS–fast and the best fixed value of T for the given instance. Although on some instances the version having fixed T performs still better, the difference in CPU seconds is not so evident like in Figure A.29 when comparing with RLS.

A.4 Robust Tabu Search

In this section we compare a Robust Tabu Search [60] (Ro–TS) approach with RLS–fast. In Ro–TS the tabu tenure is adapted by selecting throughout the search values randomly within an interval $[\mathsf{T}_{\mathsf{min}}, \mathsf{T}_{\mathsf{max}}]$. In [60] the value is updated every $2\mathsf{T}_{\mathsf{max}}$ steps, in our implementation the tenure is initialised to $T = 2$ and updated at every restart selecting uniformly randomly an integer between $\mathsf{T}_{\mathsf{min}} = 0.375|\mathsf{Best}|$ and $\mathsf{T}_{\mathsf{max}} = 0.625|\mathsf{Best}|$.

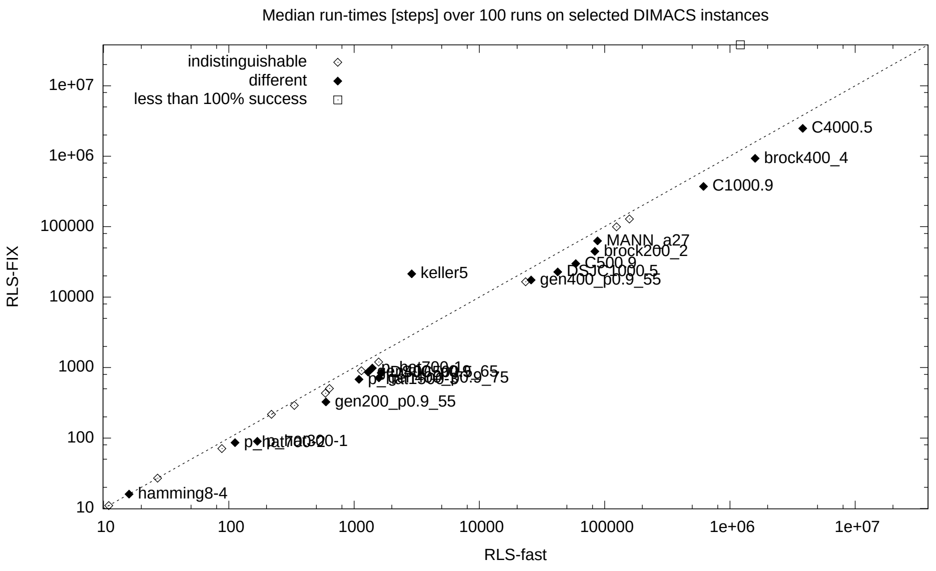


Figure A.33: Median number of steps to converge to the optimal solution of RLS with best fixed T for the instance and RLS–fast. The empty dots represent the instances for which a Mann-Whitney U-test could not reject the null hypothesis of the algorithms having an identical performance at significance level $\alpha = 0.05$. The white squares on the axes show instances for which the one of the algorithm was not able to find the maximum clique every run.

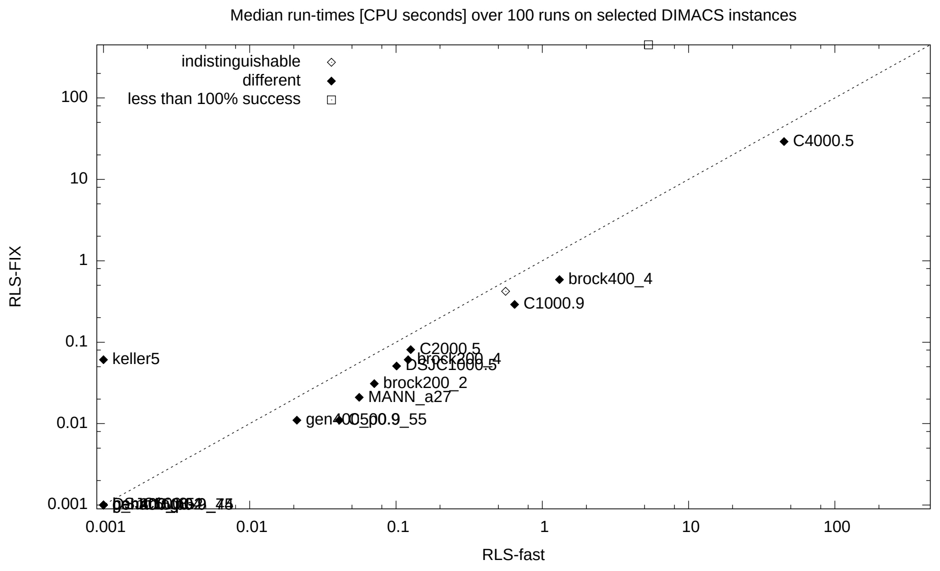


Figure A.34: Median number of CPU seconds to converge to the optimal solution of RLS with best fixed T for the instance and RLS–fast. The empty dots represent the instances for which a Mann-Whitney U-test could not reject the null hypothesis of the algorithms having an identical performance at significance level $\alpha = 0.05$. The white squares on the axes show instances for which the one of the algorithm was not able to find the maximum clique every run.

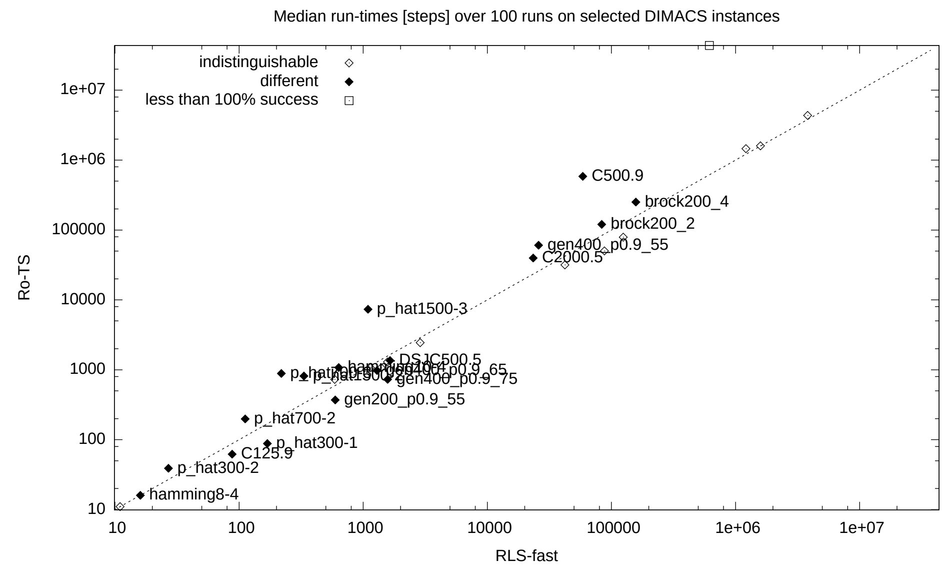


Figure A.35: Median number of steps to converge to the optimal solution of RLS–fast and Ro–TS. The empty dots represent the instances for which a Mann-Whitney U-test could not reject the null hypothesis of the algorithms having an identical performance at significance level $\alpha = 0.05$. The white squares on the axes show instances for which the one of the algorithm was not able to find the maximum clique every run.

The uniform distribution is centred around $\mu = 0.5|\mathsf{Best}|$ and its support ranges from 0.75μ to 1.25μ.

This range is narrower than the previous algorithm tested in Section A.1, namely 'random $T \in [1, 0.5(|\mathsf{Best}| + 1)]$'; and the update of the tenure is not performed at every iteration but just during the restarts.

The range $[\mathsf{T}_{\mathsf{min}}, \mathsf{T}_{\mathsf{max}}]$ seem to be appropriate for the DIMACS benchmark instances. In fact, keeping the restart frequency fixed to $R = 100$, the best tabu tenure for most instances lies around $0.5 * |\mathsf{Optimum}|$, or $0.33 * |\mathsf{Optimum}|$, or $0.24 * |\mathsf{Optimum}|$ depending from the instance family.

Figure A.35 and A.36 show the performance of Ro–TS compared to RLS–fast. Ro–TS performances are slightly worse especially considering that on two instances is not able to find the optimum solution in 100,000,000 iterations. Table A.4 shows the detailed comparison.

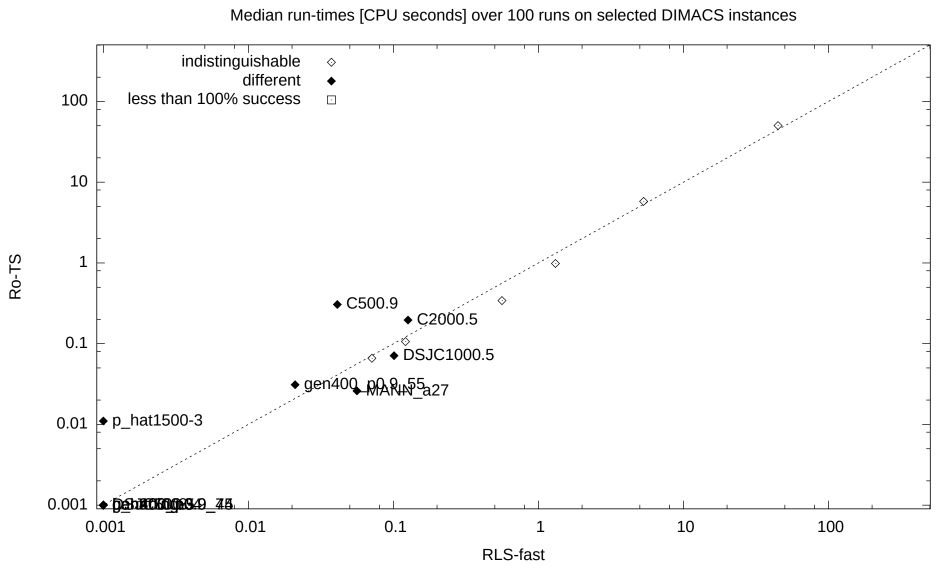


Figure A.36: Median number of CPU seconds to converge to the optimal solution of RLS–fast and Ro–TS. The empty dots represent the instances for which a Mann-Whitney U-test could not reject the null hypothesis of the algorithms having an identical performance at significance level $\alpha = 0.05$. The white squares on the axes show instances for which the one of the algorithm was not able to find the maximum clique every run.

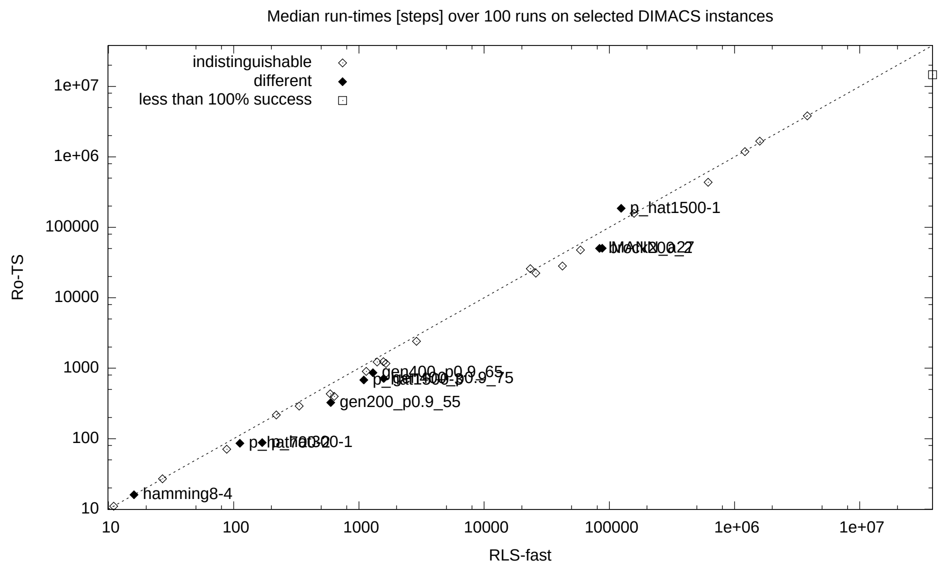


Figure A.37: Median number of steps to converge to the optimal solution of RLS–fast and Ro–TS selecting a tenure randomly around the best value for the specific instance. The empty dots represent the instances for which a Mann-Whitney U-test could not reject the null hypothesis of the algorithms having an identical performance at significance level $\alpha = 0.05$. The white squares on the axes show instances for which the one of the algorithm was not able to find the maximum clique every run.

In order to check if the worse performances of RoTS depend on the selected interval for the tabu tenure we try with $[\mathsf{T}_{\mathsf{min}}, \mathsf{T}_{\mathsf{max}}]$ where $\mathsf{T}_{\mathsf{min}} = 0.375\,\mathsf{fixT}$, $\mathsf{T}_{\mathsf{max}} = 0.625\,\mathsf{fixT}$, and fixT is the best value for the specific instance.

Figure A.37 and A.38 show how in this case Ro–TS performances are comparable or better than RLS–fast. The picture does not change if instead of the best value for the specific instance the algorithm selects randomly around a good value of T for the instance family. Figure A.39 and A.40 show the performances of a Ro–TS selecting randomly values around $0.5|\mathsf{Optimum}|$ for brock$n$_k, keller$k$, gen$n$_p0.p_k instances; $0.33|\mathsf{Optimum}|$ for MANN_an and phat$n$-k instances; $0.25|\mathsf{Optimum}|$ for Cn instances; $0.2|\mathsf{Optimum}|$ for DSJC$n.p$ and hamming$n$-d instances.

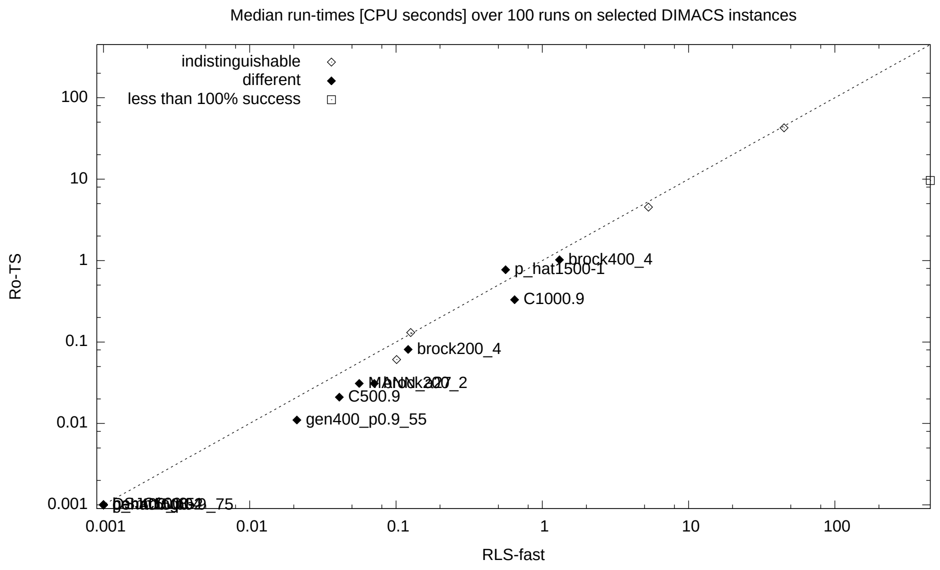


Figure A.38: Median number of CPU seconds to converge to the optimal solution of RLS–fast and Ro–TS selecting a tenure randomly around the best value for the specific instance. The empty dots represent the instances for which a Mann-Whitney U-test could not reject the null hypothesis of the algorithms having an identical performance at significance level $\alpha = 0.05$. The white squares on the axes show instances for which the one of the algorithm was not able to find the maximum clique every run.

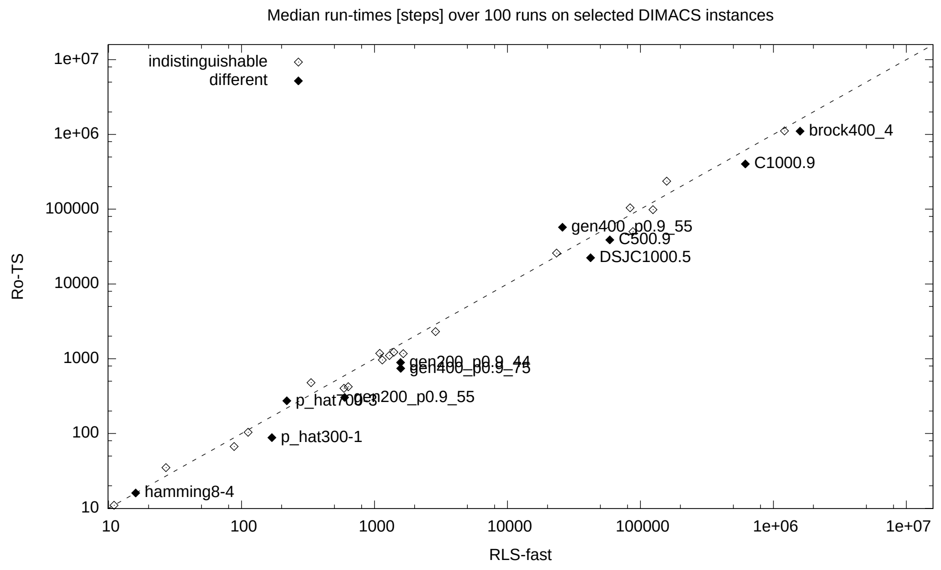


Figure A.39: Median number of steps to converge to the optimal solution of RLS–fast and Ro–TS selecting a tenure randomly around a good value for the instance family. The empty dots represent the instances for which a Mann-Whitney U-test could not reject the null hypothesis of the algorithms having an identical performance at significance level $\alpha = 0.05$. The white squares on the axes show instances for which the one of the algorithm was not able to find the maximum clique every run.

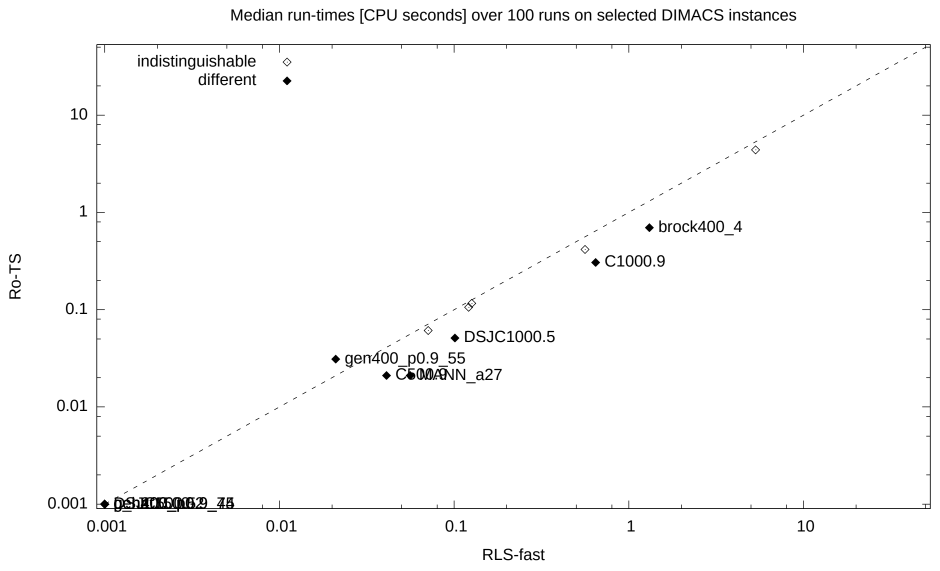


Figure A.40: Median number of CPU seconds to converge to the optimal solution of RLS–fast and Ro–TS selecting a tenure randomly around a good value for the instance family. The empty dots represent the instances for which a Mann-Whitney U-test could not reject the null hypothesis of the algorithms having an identical performance at significance level $\alpha = 0.05$. The white squares on the axes show instances for which the one of the algorithm was not able to find the maximum clique every run.

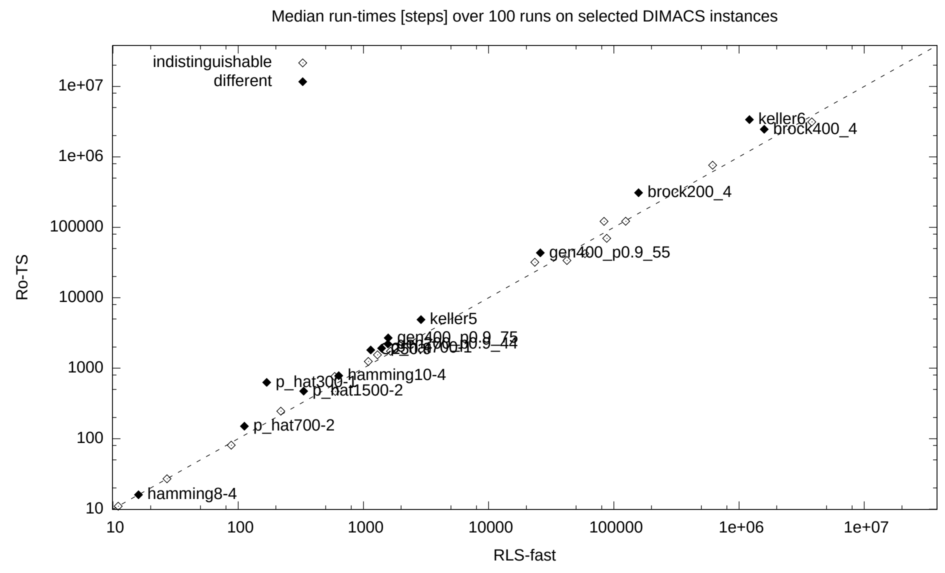


Figure A.41: Median number of steps to converge to the optimal solution of RLS–fast and Ro–TS selecting a tenure with the same distribution of RLS for the specific instance. The empty dots represent the instances for which a Mann-Whitney U-test could not reject the null hypothesis of the algorithms having an identical performance at significance level $\alpha = 0.05$. The white squares on the axes show instances for which the one of the algorithm was not able to find the maximum clique every run.

Figure A.41 and A.42 compare the performances of RLS-fix with a Ro–TS that selects a tabu tenure randomly with the same distribution of RLS for the specific instance.

A.5 Conclusions

RLS [13] is able to adapt the tabu tenure T to a value which is among the bests for the instance at hand. The only noticeable exceptions are small instances in which without a frequent restart policy the tenure T would explode. This happens in the case of small easy instances and small hard instances. In the first case the explosion would go unnoticed because of the small number of steps to find the optimum solution. In the second one

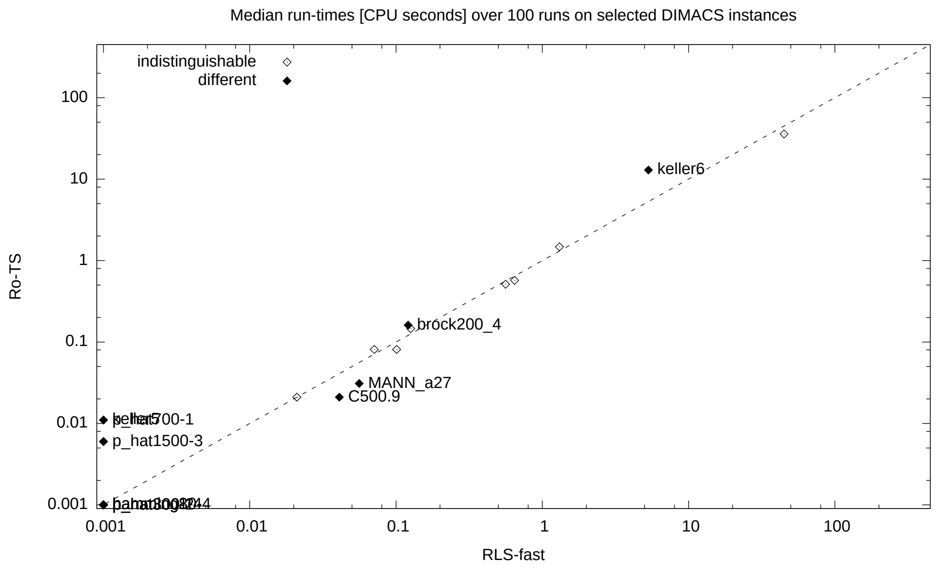


Figure A.42: Median number of CPU seconds to converge to the optimal solution of RLS–fast and Ro–TS selecting a tenure with the same distribution of RLS for the specific instance. The empty dots represent the instances for which a Mann-Whitney U-test could not reject the null hypothesis of the algorithms having an identical performance at significance level $\alpha = 0.05$. The white squares on the axes show instances for which the one of the algorithm was not able to find the maximum clique every run.

because of the frequent restarts.

Looking at the best possible diversification strategy, it seems that on the DIMACS benchmark sets spotting the right value of T leads to the best performances regardless of the restart frequency, with the only notable exception of MANN instances.

A robust tabu search centred around a correct setting for T performs as well as RLS. This result implies that at least on the DIMACS benchmark instances, there is no measurable effect showing that RLS effectively reacts to the local characteristic of the search space.

The speedup in RLS–LTM can be ascribed to the faster restarts that do not need to clear the search history. Avoiding to clear the search history is one of the causes for the explosion of the tabu tenure parameter. Another cause is the bias in the node selection during the restarts. The fact that RLS–LTM also never resets the tabu tenure during the restarts makes the effect more noticeable.

Knowing the reason for the speedup and for the explosion of the tabu tenure, we implement RLS–fast, which is fast as RLS–LTM and does not need for an upper bound for the tabu tenure.

The performances of RLS–fast are comparable to RLS–fix an algorithm that knows a priori the best tabu tenure for every instance in the DIMACS benchmark. This result shows how RLS is able to quickly converge to the best tabu tenure for the instance at hand with very little overhead.

Instance	**Best**	**reactive**			**random**		
		Solution quality	CPU(s)	Steps	Solution quality	CPU(s)	Steps
C125.9	34	34 (0.00)	0.001 (0.000)	98 (141)	34 (0.00)	0.001 (0.000)	60 (50)
C250.9	44	44 (0.00)	0.001 (0.000)	1 031 (1 124)	44 (0.00)	0.001 (0.000)	1 243 (1 842)
C500.9	57	57 (0.00)	0.061 (0.072)	87 720 (123 830)	57 (0.00)	0.021 (0.040)	30 930 (60 320)
C1000.9	68	68 (0.00)	0.741 (1.115)	756 600 (1 144 600)	68 (0.00)	0.396 (0.512)	482 600 (623 000)
C2000.9	78	78 (0.00)	-	-	78 (0.00)	-	-
DSJC1000.5	15	15 (0.00)	0.061 (0.090)	23 090 (37 950)	15 (0.00)	0.061 (0.107)	24 530 (46 164)
DSJC500.5	13	13 (0.00)	0.001 (0.000)	1 201 (1 454)	13 (0.00)	0.001 (0.000)	1 235 (1 906)
C2000.5	16	16 (0.00)	0.171 (0.253)	32 170 (48 740)	16 (0.00)	0.091 (0.110)	19 240 (23 191)
C4000.5	18	18 (0.00)	47.010 (62.910)	3 956 000 (5 249 000)	18 (0.00)	28.280 (38.960)	2 449 000 (3 407 000)
MANN_a27	126	126 (0.00)	0.036 (0.042)	62 760 (72 060)	126 (0.00)	0.041 (0.040)	75 290 (100 050)
MANN_a45	345	344 (0.29)	-	-	344 (0.29)	-	-
MANN_a81	1099	1 098 (0.09)	-	-	1 098 (0.09)	-	-
brock200_2	12	12 (0.00)	0.071 (0.102)	88 790 (129 660)	12 (0.00)	0.761 (1.292)	1 212 000 (2 035 600)
brock200_4	17	17 (0.00)	0.141 (0.265)	233 100 (406 700)	17 (0.00)	1.581 (1.795)	2 936 000 (3 337 000)
brock400_2	29	29 (0.00)	-	-	25 (16.00)	-	-
brock400_4	33	33 (0.00)	1.406 (2.660)	1 768 000 (3 223 300)	33 (0.00)	-	-
brock800_2	24	21 (14.29)	-	-	21 (14.29)	-	-
brock800_4	26	21 (23.81)	-	-	21 (23.81)	-	-
gen200_p0.9_44	44	44 (0.00)	0.001 (0.000)	1 425 (2 204)	44 (0.00)	0.001 (0.000)	1 630 (4 298)
gen200_p0.9_55	55	55 (0.00)	0.001 (0.000)	613 (668)	55 (0.00)	0.001 (0.000)	741 (1 275)
gen400_p0.9_55	55	55 (0.00)	0.016 (0.020)	22 270 (37 950)	55 (0.00)	0.021 (0.013)	35 460 (38 500)
gen400_p0.9_65	65	65 (0.00)	0.001 (0.000)	1 574 (1 601)	65 (0.00)	0.001 (0.000)	1 351 (1 852)
gen400_p0.9_75	75	75 (0.00)	0.001 (0.000)	1 614 (1 285)	75 (0.00)	0.001 (0.000)	1 573 (2 458)
hamming8-4	16	16 (0.00)	0.001 (0.000)	16 (0)	16 (0.00)	0.001 (0.000)	16 (0)
hamming10-4	40	40 (0.00)	0.001 (0.000)	529 (1 044)	40 (0.00)	0.001 (0.000)	668 (1 090)
keller4	11	11 (0.00)	0.001 (0.000)	11 (14)	11 (0.00)	0.001 (0.000)	11 (10)
keller5	27	27 (0.00)	0.001 (0.010)	2 007 (4 458)	27 (0.00)	0.011 (0.020)	10 070 (16 956)
keller6	59	59 (0.00)	2.501 (4.556)	594 900 (1 111 900)	59 (0.00)	3.726 (6.910)	957 500 (1 757 500)
p_hat300-1	8	8 (0.00)	0.001 (0.000)	102 (164)	8 (0.00)	0.001 (0.000)	128 (246)
p_hat300-2	25	25 (0.00)	0.001 (0.000)	29 (24)	25 (0.00)	0.001 (0.000)	33 (20)
p_hat300-3	36	36 (0.00)	0.001 (0.000)	418 (695)	36 (0.00)	0.001 (0.000)	809 (2 136)
p_hat700-1	11	11 (0.00)	0.001 (0.010)	725 (1 636)	11 (0.00)	0.001 (0.010)	1 243 (1 939)
p_hat700-2	44	44 (0.00)	0.001 (0.000)	112 (83)	44 (0.00)	0.001 (0.000)	102 (80)
p_hat700-3	62	62 (0.00)	0.001 (0.000)	212 (273)	62 (0.00)	0.001 (0.000)	205 (238)
p_hat1500-1	12	12 (0.00)	0.501 (0.800)	113 600 (189 520)	12 (0.00)	0.671 (0.971)	164 100 (244 940)
p_hat1500-2	65	65 (0.00)	0.001 (0.010)	250 (995)	65 (0.00)	0.001 (0.010)	262 (709)
p_hat1500-3	94	94 (0.00)	0.001 (0.010)	1 253 (1 498)	94 (0.00)	0.001 (0.010)	685 (1 004)

Table A.1: Algorithm comparison on a selected sub-set of the DIMACS benchmark instances. Threshold MAX_T is set to $0.5(|\mathsf{Best}|+1)$. The table shows the median solution quality and within brackets the percentage deviation from the best known, as well as CPU seconds and steps medians with IQR within brackets.

Instance	Best	reactive			random		
		Solution quality	CPU(s)	Steps	Solution quality	CPU(s)	Steps
C125.9	34	34 (0.00)	0.001 (0.000)	88 (150)	34 (0.00)	0.001 (0.000)	361 (649)
C250.9	44	44 (0.00)	0.001 (0.000)	1 068 (1 039)	44 (0.00)	0.001 (0.010)	9 788 (14 142)
C500.9	57	57 (0.00)	0.056 (0.125)	82 740 (181 010)	57 (0.00)	-	-
C1000.9	68	68 (0.00)	-	-	66 (3.03)	-	-
C2000.9	78	78 (0.00)	-	-	74 (5.41)	-	-
DSJC1000.5	15	15 (0.00)	0.086 (0.132)	34 880 (54 750)	15 (0.00)	0.261 (0.483)	105 400 (196 390)
DSJC500.5	13	13 (0.00)	0.001 (0.010)	1 629 (2 227)	13 (0.00)	0.011 (0.010)	4 358 (8 575)
C2000.5	16	16 (0.00)	0.176 (0.243)	33 080 (45 190)	16 (0.00)	0.506 (0.810)	96 700 (158 840)
C4000.5	18	17 (5.88)	-	-	18 (0.00)	-	-
MANN_a27	126	126 (0.00)	0.041 (0.040)	62 760 (72 060)	126 (0.00)	0.011 (0.030)	27 450 (42 090)
MANN_a45	345	344 (0.29)	-	-	344 (0.29)	-	-
MANN_a81	1099	1 098 (0.09)	-	-	1 098 (0.09)	-	-
brock200_2	12	12 (0.00)	0.071 (0.133)	68 840 (126 410)	12 (0.00)	0.041 (0.050)	60 150 (77 500)
brock200_4	17	17 (0.00)	1.161 (2.045)	1 335 000 (2 334 800)	17 (0.00)	0.091 (0.152)	172 700 (292 020)
brock400_2	29	25 (16.00)	-	-	29 (0.00)	-	-
brock400_4	33	32 (3.12)	-	-	33 (0.00)	1.246 (2.250)	1 712 000 (3 099 000)
brock800_2	24	21 (14.29)	-	-	21 (14.29)	-	-
brock800_4	26	21 (23.81)	-	-	21 (23.81)	-	-
gen200_p0.9_44	44	44 (0.00)	0.001 (0.000)	1 556 (1 906)	44 (0.00)	0.011 (0.010)	13 310 (20 423)
gen200_p0.9_55	55	55 (0.00)	0.001 (0.000)	596 (562)	55 (0.00)	0.001 (0.000)	2 685 (6 081)
gen400_p0.9_55	55	55 (0.00)	0.011 (0.010)	21 530 (28 390)	55 (0.00)	1.896 (3.725)	3 440 000 (6 820 000)
gen400_p0.9_65	65	65 (0.00)	0.001 (0.000)	1 294 (1 132)	65 (0.00)	0.021 (0.030)	34 210 (64 630)
gen400_p0.9_75	75	75 (0.00)	0.001 (0.000)	1 576 (1 184)	75 (0.00)	0.021 (0.030)	33 870 (63 110)
hamming8-4	16	16 (0.00)	0.001 (0.000)	16 (0)	16 (0.00)	0.001 (0.000)	16 (0)
hamming10-4	40	40 (0.00)	0.001 (0.000)	529 (1 044)	40 (0.00)	0.016 (0.020)	11 220 (15 669)
keller4	11	11 (0.00)	0.001 (0.000)	11 (4)	11 (0.00)	0.001 (0.000)	11 (22)
keller5	27	27 (0.00)	0.001 (0.010)	2 892 (5 699)	27 (0.00)	0.011 (0.020)	7 538 (12 229)
keller6	59	59 (0.00)	-	-	56 (5.36)	-	-
p_hat300-1	8	8 (0.00)	0.001 (0.000)	138 (278)	8 (0.00)	0.001 (0.000)	160 (261)
p_hat300-2	25	25 (0.00)	0.001 (0.000)	27 (20)	25 (0.00)	0.001 (0.000)	38 (242)
p_hat300-3	36	36 (0.00)	0.001 (0.000)	633 (1 232)	36 (0.00)	0.001 (0.010)	6 956 (11 956)
p_hat700-1	11	11 (0.00)	0.001 (0.010)	1 143 (2 864)	11 (0.00)	0.001 (0.010)	1 877 (3 104)
p_hat700-2	44	44 (0.00)	0.001 (0.000)	112 (83)	44 (0.00)	0.001 (0.010)	2 394 (3 892)
p_hat700-3	62	62 (0.00)	0.001 (0.000)	219 (282)	62 (0.00)	0.011 (0.010)	15 450 (24 640)
p_hat1500-1	12	12 (0.00)	3.326 (7.451)	672 800 (1 475 200)	12 (0.00)	1.086 (1.586)	255 400 (364 100)
p_hat1500-2	65	65 (0.00)	0.001 (0.010)	331 (1 244)	65 (0.00)	0.041 (0.062)	18 880 (33 812)
p_hat1500-3	94	94 (0.00)	0.001 (0.010)	1 253 (1 498)	94 (0.00)	0.766 (0.967)	536 500 (659 300)

Table A.2: Algorithm comparison on a selected sub-set of the DIMACS benchmark instances. Threshold MAX_T is set to 2.0(|Best|+1). The table shows the median solution quality and within brackets the percentage deviation from the best known, as well as CPU seconds and steps medians with IQR within brackets.

Instance	Best	RLS-LTM			RLS		
		Solution quality	CPU(s)	Steps	Solution quality	CPU(s)	Steps
C125.9	34	34 (0.00)	0.001 (0.000)	98 (141)	34 (0.00)	0.001 (0.000)	88 (150)
C250.9	44	44 (0.00)	0.001 (0.000)	1 031 (1 124)	44 (0.00)	0.001 (0.000)	1 192 (1 081)
C500.9	57	57 (0.00)	0.061 (0.072)	87 720 (123 830)	57 (0.00)	2.471 (3.570)	67 890 (92 280)
C1000.9	68	68 (0.00)	0.741 (1.115)	756 600 (1 144 600)	68 (0.00)	10.780 (15.051)	763 400 (1 052 200)
C2000.9	78	78 (0.00)	-	-	78 (0.00)	-	-
DSJC1000.5	15	15 (0.00)	0.061 (0.090)	23 090 (37 950)	15 (0.00)	6.291 (7.709)	40 930 (48 660)
DSJC500.5	13	13 (0.00)	0.001 (0.000)	1 201 (1 454)	13 (0.00)	0.221 (0.430)	1 681 (1 843)
C2000.5	16	16 (0.00)	0.171 (0.253)	32 170 (48 740)	16 (0.00)	4.096 (6.685)	27 160 (42 750)
C4000.5	18	18 (0.00)	47.010 (62.910)	3 956 000 (5 249 000)	18 (0.00)	342.600 (512.300)	3 556 000 (5 209 000)
MANN_a27	126	126 (0.00)	0.036 (0.042)	62 760 (72 060)	126 (0.00)	1.191 (1.725)	62 760 (90 650)
MANN_a45	345	344 (0.29)	-	-	344 (0.29)	-	-
MANN_a81	1099	1 098 (0.09)	-	-	1 098 (0.09)	-	-
brock200_2	12	12 (0.00)	0.071 (0.102)	88 790 (129 660)	12 (0.00)	8.206 (20.069)	42 370 (106 090)
brock200_4	17	17 (0.00)	0.141 (0.265)	233 100 (406 700)	17 (0.00)	29.120 (52.620)	228 300 (356 500)
brock400_2	29	29 (0.00)	-	-	29 (0.00)	-	-
brock400_4	33	33 (0.00)	1.406 (2.660)	1 768 000 (3 223 300)	33 (0.00)	44.240 (70.090)	1 741 000 (2 749 500)
brock800_2	24	21 (14.29)	-	-	21 (14.29)	-	-
brock800_4	26	21 (23.81)	-	-	21 (23.81)	-	-
gen200_p0.9_44	44	44 (0.00)	0.001 (0.000)	1 425 (2 204)	44 (0.00)	0.001 (0.000)	1 393 (1 982)
gen200_p0.9_55	55	55 (0.00)	0.001 (0.000)	613 (668)	55 (0.00)	0.001 (0.000)	596 (562)
gen400_p0.9_55	55	55 (0.00)	0.016 (0.020)	22 270 (37 950)	55 (0.00)	0.711 (1.245)	21 540 (30 572)
gen400_p0.9_65	65	65 (0.00)	0.001 (0.000)	1 574 (1 601)	65 (0.00)	0.001 (0.000)	1 294 (1 132)
gen400_p0.9_75	75	75 (0.00)	0.001 (0.000)	1 614 (1 285)	75 (0.00)	0.001 (0.000)	1 576 (1 184)
hamming8-4	16	16 (0.00)	0.001 (0.000)	16 (0)	16 (0.00)	0.001 (0.000)	16 (0)
hamming10-4	40	40 (0.00)	0.001 (0.000)	529 (1 044)	40 (0.00)	0.001 (0.010)	527 (848)
keller4	11	11 (0.00)	0.001 (0.000)	11 (14)	11 (0.00)	0.001 (0.000)	11 (4)
keller5	27	27 (0.00)	0.001 (0.010)	2 007 (4 458)	27 (0.00)	0.011 (0.250)	2 727 (4 976)
keller6	59	59 (0.00)	2.501 (4.556)	594 900 (1 111 900)	59 (0.00)	42.130 (86.100)	1 123 000 (2 064 900)
p_hat300-1	8	8 (0.00)	0.001 (0.000)	102 (164)	8 (0.00)	0.001 (0.000)	116 (227)
p_hat300-2	25	25 (0.00)	0.001 (0.000)	29 (24)	25 (0.00)	0.001 (0.000)	27 (20)
p_hat300-3	36	36 (0.00)	0.001 (0.000)	418 (695)	36 (0.00)	0.001 (0.000)	560 (879)
p_hat700-1	11	11 (0.00)	0.001 (0.010)	725 (1 636)	11 (0.00)	0.201 (0.412)	1 298 (1 938)
p_hat700-2	44	44 (0.00)	0.001 (0.000)	112 (83)	44 (0.00)	0.001 (0.000)	112 (83)
p_hat700-3	62	62 (0.00)	0.001 (0.000)	212 (273)	62 (0.00)	0.001 (0.000)	219 (282)
p_hat1500-1	12	12 (0.00)	0.501 (0.800)	113 600 (189 520)	12 (0.00)	33.320 (49.930)	157 800 (226 250)
p_hat1500-2	65	65 (0.00)	0.001 (0.010)	250 (995)	65 (0.00)	0.001 (0.010)	340 (1 288)
p_hat1500-3	94	94 (0.00)	0.001 (0.010)	1 253 (1 498)	94 (0.00)	0.006 (0.010)	1 189 (1 699)

Table A.3: Algorithm comparison on a selected sub-set of the DIMACS benchmark instances. The table shows the median solution quality and within brackets the percentage deviation from the best known, as well as CPU seconds and steps medians with IQR within brackets.

Instance	Best	Ro-TS			RLS-fast		
		Solution quality	CPU(s)	Steps	Solution quality	CPU(s)	Steps
C125.9	34	34 (0.00)	0.001 (0.000)	62 (62)	34 (0.00)	0.001 (0.000)	88 (150)
C250.9	44	44 (0.00)	0.001 (0.000)	1 071 (1 663)	44 (0.00)	0.001 (0.000)	1 144 (1 031)
C500.9	57	57 (0.00)	0.306 (0.410)	586 100 (777 600)	57 (0.00)	0.041 (0.060)	58 820 (72 930)
C1000.9	68	68 (0.00)	-	-	68 (0.00)	0.646 (1.210)	615 600 (1 176 900)
C2000.9	78	76 (2.63)	-	-	78 (0.00)	-	-
DSJC1000.5	15	15 (0.00)	0.071 (0.113)	31 730 (48 018)	15 (0.00)	0.101 (0.172)	42 230 (68 900)
DSJC500.5	13	13 (0.00)	0.001 (0.000)	1 353 (1 762)	13 (0.00)	0.001 (0.010)	1 646 (2 104)
C2000.5	16	16 (0.00)	0.196 (0.257)	39 760 (52 130)	16 (0.00)	0.126 (0.233)	23 400 (43 868)
C4000.5	18	18 (0.00)	50.370 (77.500)	4 364 000 (6 713 000)	18 (0.00)	44.890 (58.050)	3 813 000 (4 936 000)
MANN_a27	126	126 (0.00)	0.026 (0.040)	50 270 (112 550)	126 (0.00)	0.056 (0.073)	87 760 (125 040)
MANN_a45	345	344 (0.29)	-	-	344 (0.29)	-	-
MANN_a81	1099	1 098 (0.09)	-	-	1 098 (0.09)	-	-
brock200_2	12	12 (0.00)	0.066 (0.112)	120 300 (197 660)	12 (0.00)	0.071 (0.103)	83 580 (139 510)
brock200_4	17	17 (0.00)	0.106 (0.190)	251 100 (431 790)	17 (0.00)	0.121 (0.180)	157 600 (247 080)
brock400_2	29	29 (0.00)	-	-	29 (0.00)	-	-
brock400_4	33	33 (0.00)	0.986 (1.535)	1 600 000 (2 369 600)	33 (0.00)	1.311 (2.178)	1 588 000 (2 618 900)
brock800_2	24	21 (14.29)	-	-	21 (14.29)	-	-
brock800_4	26	21 (23.81)	-	-	21 (23.81)	-	-
gen200_p0.9_44	44	44 (0.00)	0.001 (0.000)	1 299 (1 861)	44 (0.00)	0.001 (0.000)	1 571 (1 916)
gen200_p0.9_55	55	55 (0.00)	0.001 (0.000)	370 (432)	55 (0.00)	0.001 (0.000)	596 (562)
gen400_p0.9_55	55	55 (0.00)	0.031 (0.050)	60 510 (90 150)	55 (0.00)	0.021 (0.020)	25 900 (33 820)
gen400_p0.9_65	65	65 (0.00)	0.001 (0.000)	975 (806)	65 (0.00)	0.001 (0.000)	1 294 (1 132)
gen400_p0.9_75	75	75 (0.00)	0.001 (0.000)	729 (805)	75 (0.00)	0.001 (0.000)	1 576 (1 184)
hamming8-4	16	16 (0.00)	0.001 (0.000)	16 (0)	16 (0.00)	0.001 (0.000)	16 (0)
hamming10-4	40	40 (0.00)	0.001 (0.003)	1 075 (1 556)	40 (0.00)	0.001 (0.000)	635 (1 049)
keller4	11	11 (0.00)	0.001 (0.000)	11 (4)	11 (0.00)	0.001 (0.000)	11 (12)
keller5	27	27 (0.00)	0.001 (0.010)	2 440 (4 410)	27 (0.00)	0.001 (0.010)	2 880 (4 513)
keller6	59	59 (0.00)	5.796 (11.432)	1 456 000 (2 799 400)	59 (0.00)	5.316 (9.687)	1 212 000 (2 227 600)
p_hat300-1	8	8 (0.00)	0.001 (0.000)	88 (133)	8 (0.00)	0.001 (0.000)	169 (273)
p_hat300-2	25	25 (0.00)	0.001 (0.000)	39 (48)	25 (0.00)	0.001 (0.000)	27 (20)
p_hat300-3	36	36 (0.00)	0.001 (0.000)	730 (748)	36 (0.00)	0.001 (0.000)	590 (1 036)
p_hat700-1	11	11 (0.00)	0.001 (0.010)	1 072 (1 332)	11 (0.00)	0.001 (0.010)	1 398 (1 786)
p_hat700-2	44	44 (0.00)	0.001 (0.000)	198 (270)	44 (0.00)	0.001 (0.000)	112 (83)
p_hat700-3	62	62 (0.00)	0.001 (0.000)	886 (1 095)	62 (0.00)	0.001 (0.000)	219 (282)
p_hat1500-1	12	12 (0.00)	0.341 (0.715)	78 840 (166 240)	12 (0.00)	0.561 (0.760)	124 400 (163 880)
p_hat1500-2	65	65 (0.00)	0.001 (0.010)	816 (1 401)	65 (0.00)	0.001 (0.010)	333 (1 290)
p_hat1500-3	94	94 (0.00)	0.011 (0.020)	7 321 (11 925)	94 (0.00)	0.001 (0.010)	1 095 (1 546)

Table A.4: Algorithm comparison on a selected sub-set of the DIMACS benchmark instances. The table shows the median solution quality and within brackets the percentage deviation from the best known, as well as CPU seconds and steps medians with IQR within brackets.

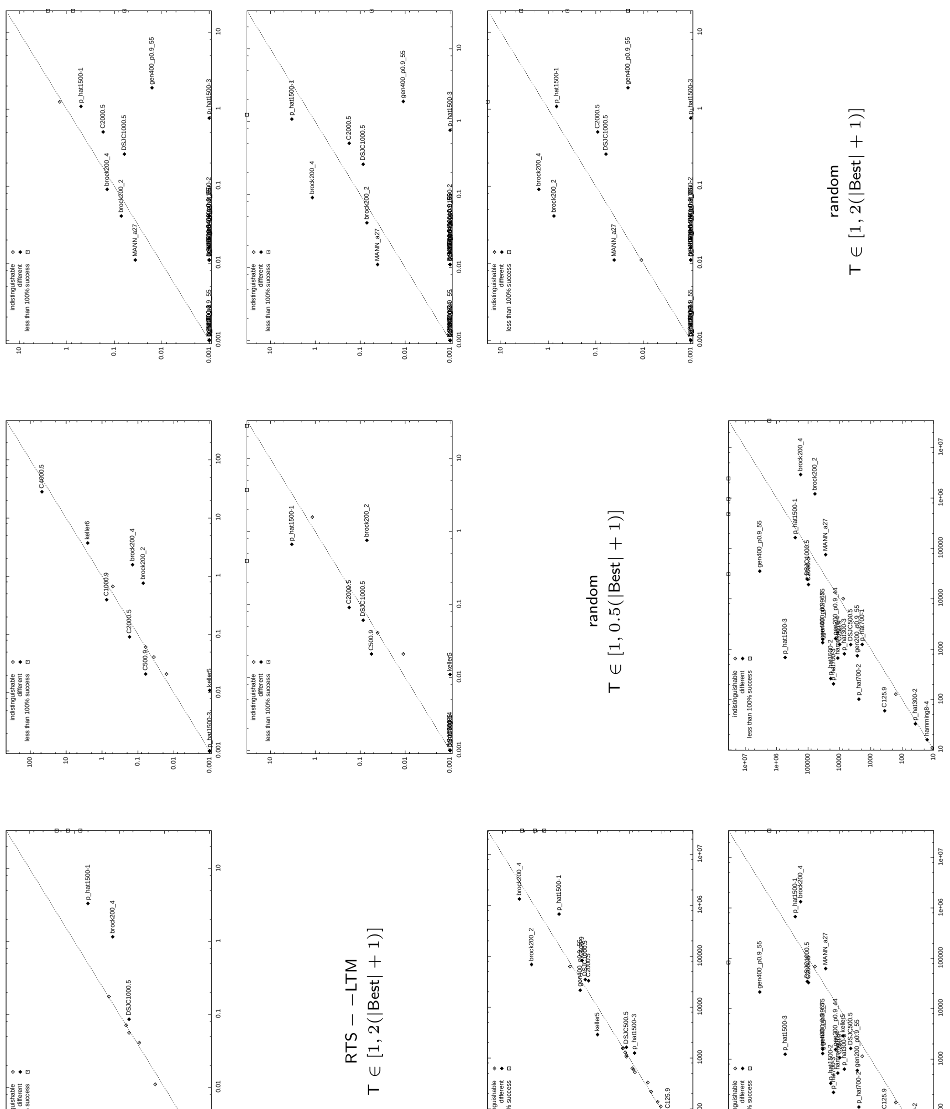


Table A.5: Median number of steps (lower triangular) and median number of CPU seconds (upper triangular) to converge to the optimal solution. The empty dots represent the instances for which a Mann-Whitney U-test could not reject the null hypothesis of the algorithms having an identical performance at significance level $\alpha = 0.05$. The white squares on the axes show instances for which the one of the algorithm was not able to find the maximum clique every run. The white squares on the axes show instances for which the one of the algorithm was not able to find the maximum clique every run.

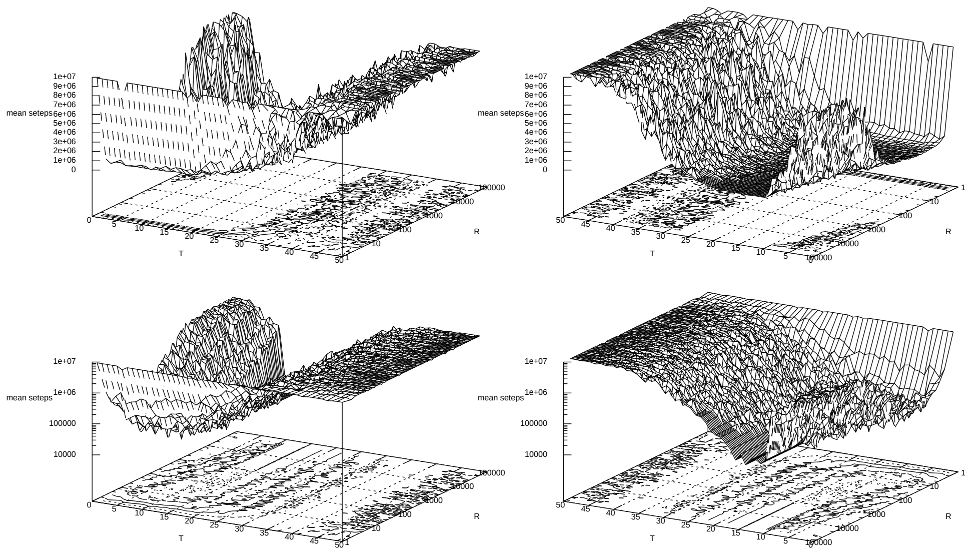


Figure A.43: Mean number of iteration to find the optimum solution of C500.9. The mean is over 10 runs of 10,000,000 iterations with fixed values for the tabu tenure and restarts frequencies. From left to right two different rotations around the y axis, and from top to bottom the z axis is in linear and log scale.

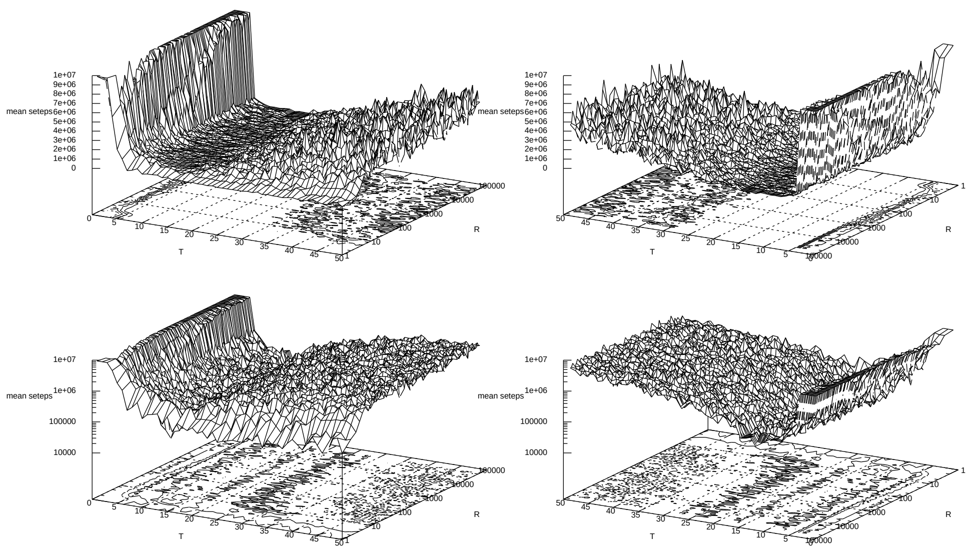


Figure A.44: Mean number of iteration to find the optimum solution of brock200_4. The mean is over 10 runs of 10,000,000 iterations with fixed values for the tabu tenure and restarts frequencies. From left to right two different rotations around the y axis, and from top to bottom the z axis is in linear and log scale.

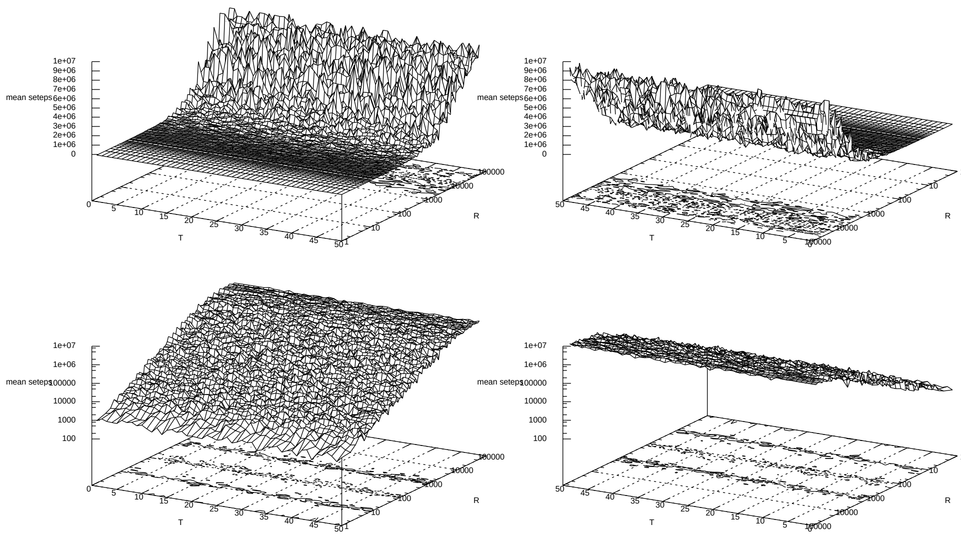


Figure A.45: Mean number of iteration to find the optimum solution of MANN_a27. The mean is over 10 runs of 10,000,000 iterations with fixed values for the tabu tenure and restarts frequencies. From left to right two different rotations around the y axis, and from top to bottom the z axis is in linear and log scale.

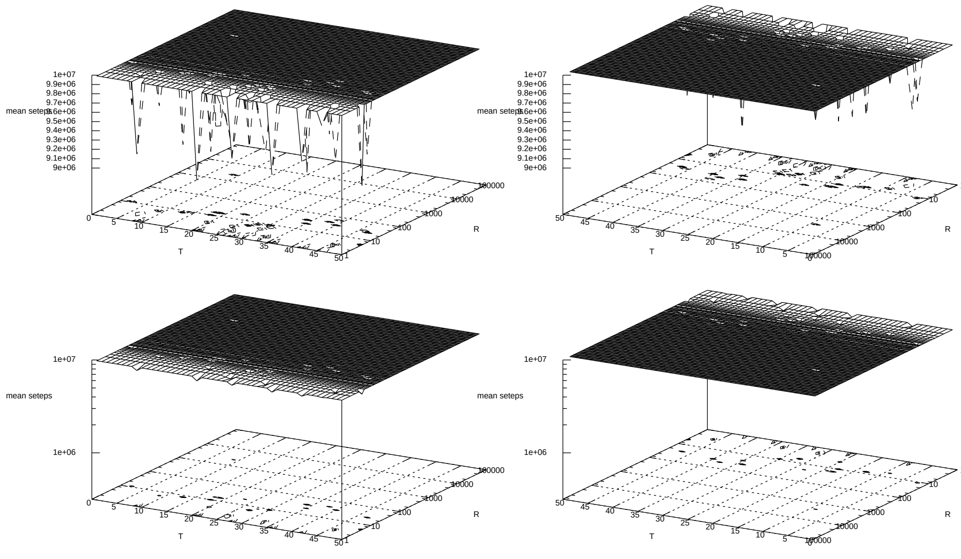


Figure A.46: Mean number of iteration to find the optimum solution of MANN_a45. The mean is over 10 runs of 10,000,000 iterations with fixed values for the tabu tenure and restarts frequencies. From left to right two different rotations around the y axis, and from top to bottom the z axis is in linear and log scale.

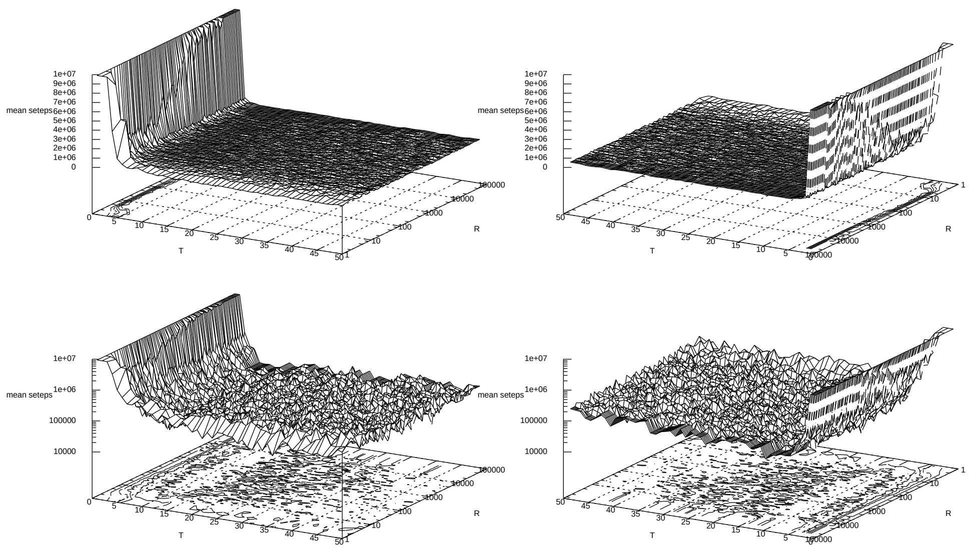


Figure A.47: Mean number of iteration to find the optimum solution of brock200_2. The mean is over 10 runs of 10,000,000 iterations with fixed values for the tabu tenure and restarts frequencies. From left to right two different rotations around the y axis, and from top to bottom the z axis is in linear and log scale.

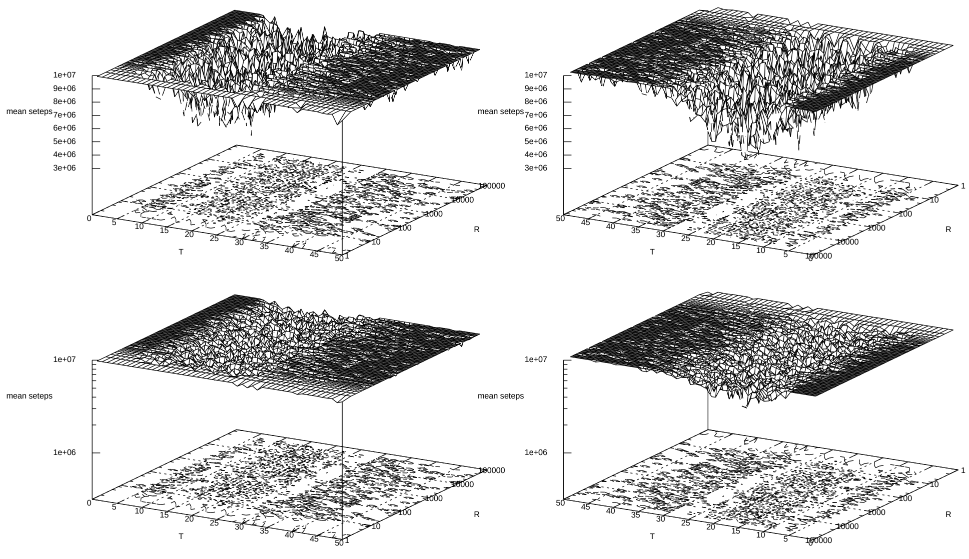


Figure A.48: Mean number of iteration to find the optimum solution of brock400_2. The mean is over 10 runs of 10,000,000 iterations with fixed values for the tabu tenure and restarts frequencies. From left to right two different rotations around the y axis, and from top to bottom the z axis is in linear and log scale.

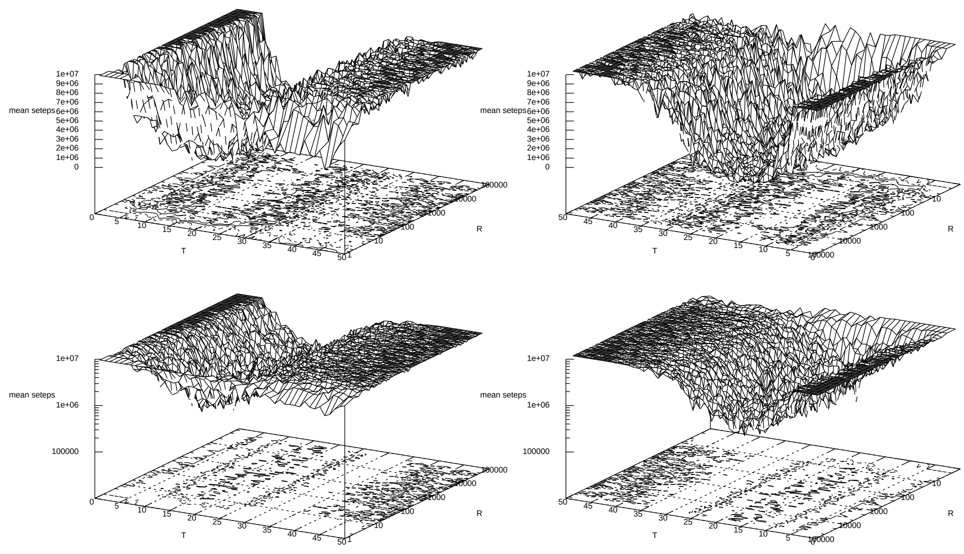


Figure A.49: Mean number of iteration to find the optimum solution of brock400_4. The mean is over 10 runs of 10,000,000 iterations with fixed values for the tabu tenure and restarts frequencies. From left to right two different rotations around the y axis, and from top to bottom the z axis is in linear and log scale.

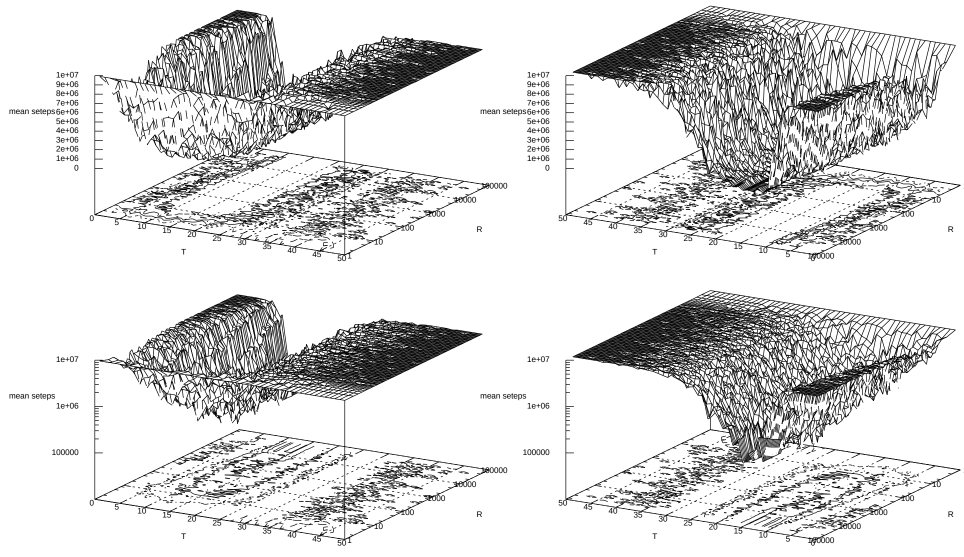


Figure A.50: Mean number of iteration to find the optimum solution of C1000.9. The mean is over 10 runs of 10,000,000 iterations with fixed values for the tabu tenure and restarts frequencies. From left to right two different rotations around the y axis, and from top to bottom the z axis is in linear and log scale.

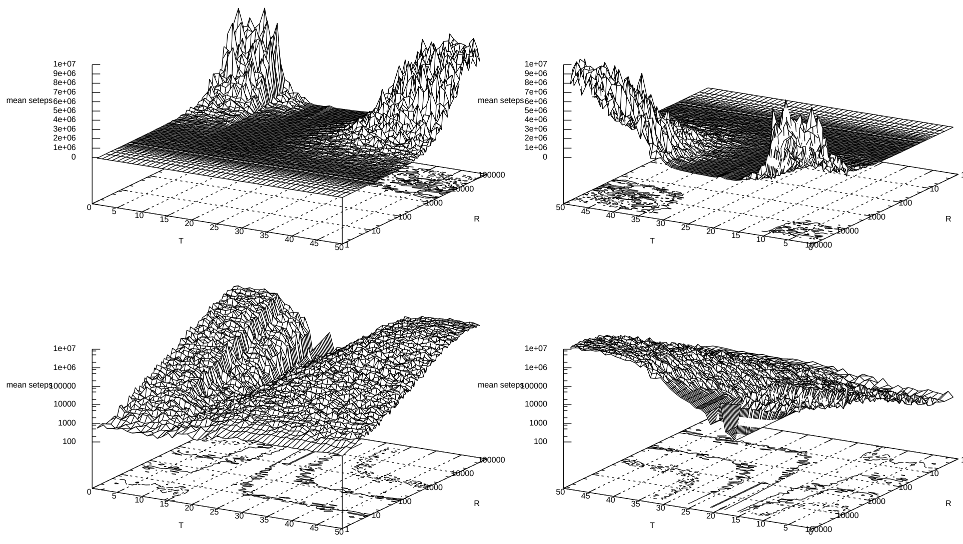


Figure A.51: Mean number of iteration to find the optimum solution of keller5. The mean is over 10 runs of 10,000,000 iterations with fixed values for the tabu tenure and restarts frequencies. From left to right two different rotations around the y axis, and from top to bottom the z axis is in linear and log scale.

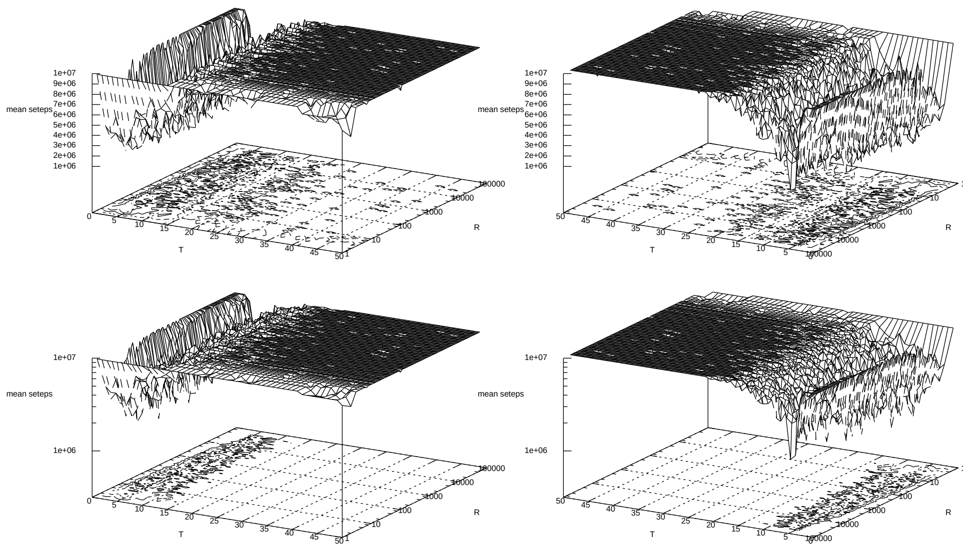


Figure A.52: Mean number of iteration to find the optimum solution of C4000.5. The mean is over 10 runs of 10,000,000 iterations with fixed values for the tabu tenure and restarts frequencies. From left to right two different rotations around the y axis, and from top to bottom the z axis is in linear and log scale.

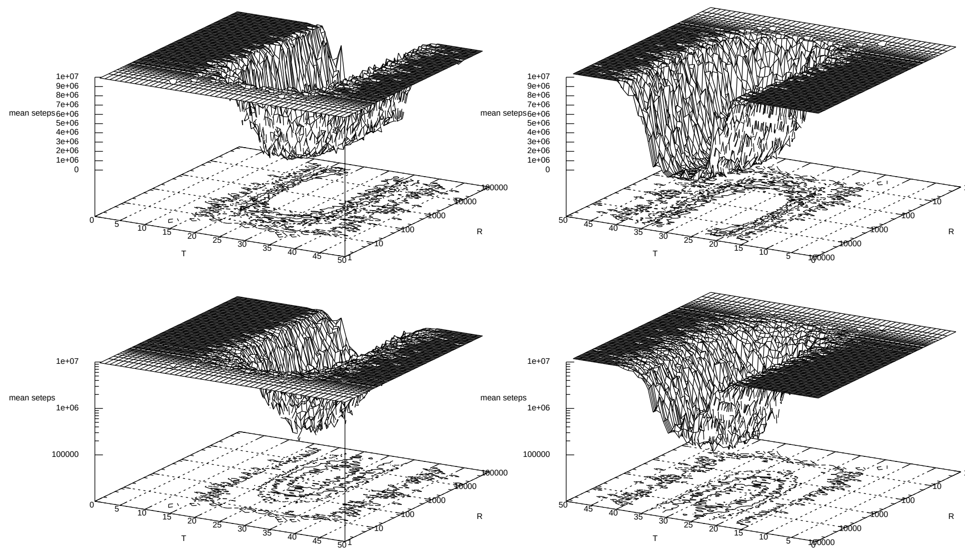


Figure A.53: Mean number of iteration to find the optimum solution of keller6. The mean is over 10 runs of 10,000,000 iterations with fixed values for the tabu tenure and restarts frequencies. From left to right two different rotations around the y axis, and from top to bottom the z axis is in linear and log scale.

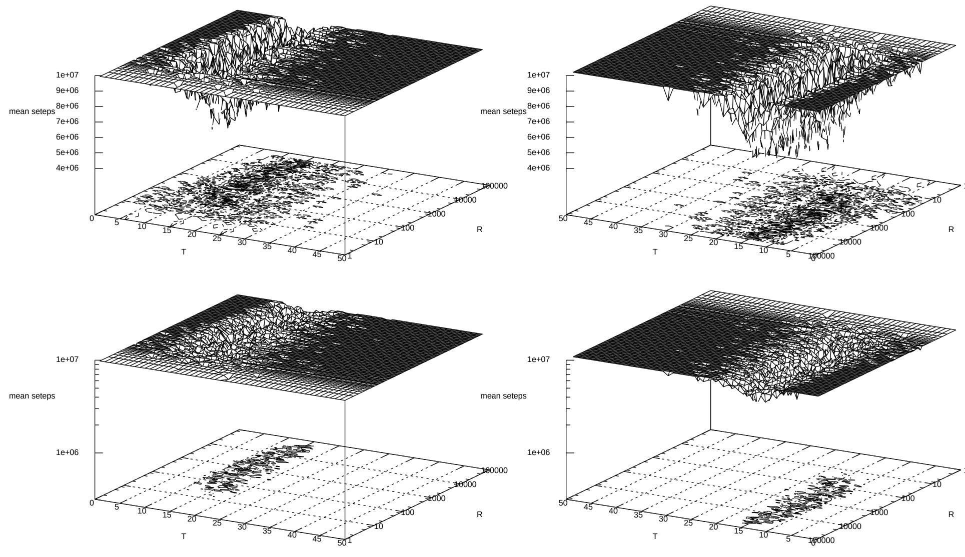


Figure A.54: Mean number of iteration to find the optimum solution of C2000.9. The mean is over 10 runs of 10,000,000 iterations with fixed values for the tabu tenure and restarts frequencies. From left to right two different rotations around the y axis, and from top to bottom the z axis is in linear and log scale.