Utrecht University

Master Thesis Applied Computing Science

INF/SCR-09-95

An analysis of tabu search for the graph coloring problem

Author:
Jan Olav Hajduk

Supervisor:
dr. ir. D. Thierens

July 16, 2010

Abstract

The graph coloring problem is about finding a way to color a given graph with the smallest number of colors without allowing adjacent vertices to have the same color. The tabu search algorithm has been used extensively for the graph coloring problem. In this paper we will investigate what influence the tabu search parameters, which control the duration in which a certain region in the search space is excluded from the search, have on the performance of the algorithm. We will then show that the number of parameters of tabu search for graph coloring can be reduced without compromising performance. However, the remaining parameters have to be chosen with care. We will propose a tabu search mechanism which can be used to determine whether the search algorithm is stuck in a region of the search space, which can be the case due to a poor choice of parameter values.

Contents

Chapter 1

Introduction

Problem definition: Given is a graph $G = (V, E)$, where V is the vertex and E the edge set. The number of colors, k is a positive integer. A coloring on this graph is a function $c : V \rightarrow (1, 2, ..., k)$. This can also be seen as a partition: a coloring $C = (c(1), c(2), ..., c(|V|))$ is partition $C^1, C^2, ..., C^k$ of V such that $\forall x \in V, x \in C^i \Leftrightarrow c(x) = i$. This can be used to avoid symmetry issues when comparing two colorings, as is done for position-guided tabu search as will be shown later.

Now the graph coloring problem (GCP) is finding the smallest k for which G is k-colorable. The $k - GCP$ problem is a subset of this problem and is about whether a graph is k-colorable. A configuration is k-colorable if $\forall (i, j) \in E$ it holds that $c(i) \neq c(j)$. A common way to find the smallest k is to pick a reasonable value for k and try to find a solution. If this succeeds we try to find a solution for $k = k - 1$. This is repeated until no solution can be found for a certain k. The smallest k for which a graph G is k-colorable is called the chromatic number of G which is denoted by $\chi(G)$.

Finding a solution. As R.M, Karp showed [18], solving the graph coloring problem is a NP-complete problem. This is why we will use a heuristic to find a solution for any problem of a reasonable size. Tabu search for graph coloring (Tabucol) is a local search algorithm which deals with this by making only a single modification (a move) at each iteration which seems most favorable at that point and repeat this until a solution is found, while avoiding solutions it has just visited to avoid getting stuck in local optima. Additionally we will take a look at Position guided tabu search, which is an extension of Tabucol using an additional escape method to avoid areas in the search space which have been visited before.

Overview: chapter 2 gives a overview of work that has been done on methods for finding solutions for the graph coloring problem. After this we will look into the experimental side of the project, determining how sensitive Tabucol is for variations in its parameters, in chapter 3. The next section in this chapter explores why certain values for the parameters result in poor performance of Tabucol. We will also see that Tabucol performs well with a limited number of parameters, assuming these are chosen with care. We also propose a tabu search mechanism for determining how the moves performed are divided over the vertices, which can be used to determine whether the search algorithm is stuck in a region of the search space. Chapter 4 contains discussion points about obtained results and points of interest for further research. Finally, the conclusion is presented in chapter 5.

Chapter 2

Background

2.1 Historical overview of algorithms for graph coloring

The first algorithms used for solving the graph coloring problem were greedy constructive algorithms. These algorithms work by successively coloring vertices. Most of these algorithms are very fast, but tend to find low quality solutions. Two well known greedy constructive algorithms algorithms are Recursive Largest First (RLF) and DSATUR.

The RLF algorithmwas proposed by Leighton in 1979 [20]. RLF colors vertices in descending order according to the vertex degree, which is defined as the number of vertices a vertex is connected to. Vertices are assigned the lowest available color class. This is repeated until all vertices either have been colored or cannot be assigned a color without introducing a conflict. The idea behind this is that this will increase the chance that difficult vertices (which are likely to have a high degree) will be assigned a color early. This will hopefully avoid situations where a vertex cannot be assigned a color, and thus increase the number of vertices colored in the end.

DSATUR, proposed by Brélaz in 1979 [4], is similar to RLF. Instead by the vertex degree, the order vertices are colored is determined by their saturation. The saturation of a vertex is defined as the number of different colors a vertex is connected to. If none of the neighboring vertices have been colored yet, it is equal to the degree of the vertex. Initially the vertex with the highest degree (number of connected vertices, as no vertices have been colored yet) is colored. Then the other vertices are colored with the lowest possible color in order of decreasing saturation degree, where ties are selected by picking the vertex with the highest degree. This method is a bit more refined than RLF as it prioritizes vertices with few available colors instead of treating all vertices with the same degree equally. Brélaz proved that for bipartite graphs the algorithm is exact, which means that it is guaranteed to find an optimal solution. For other graphs it produces good results considering its low complexity.

More recently, local search algorithms have been used extensively for the graph coloring problem [12], either as standalone, or in a hybrid form supporting some other algorithm.

The first local search algorithm for graph coloring, which was proposed by Chams et al. in 1987 [5], was a version of simulated annealing (SA) applied to graph coloring.

Starting from an initial solution, a neighboring solution is selected. These are all the solutions reachable by changing the color of a single vertex. If this neighboring solution is better than the current solution it is accepted and the algorithm starts over with this new solution as starting solution. If it is not better it will be accepted anyway with probability $p(t)$, where $p(t)$ is controlled by the parameter t which represents he 'temperature' of the search. Initially the 'temperature' is set to a high value which decreases as the search progresses. This results in a gradual decrease in probability of accepting worse solutions.

Besides experimenting with simulated annealing, Chams et al. also experimented with combined algorithms, used in two phases on the same graph. As a pre-processing step they chose the previously mentioned greedy algorithms for graph coloring like DSATUR because they are fast, yet produce reasonable results. Besides DSATUR they also used a modified version of the RLF algorithm, the RLF** algorithm. This algorithm, which was proposed by Johnson, is based on the RLF algorithm of Leighton and the GE2 method of Johri and Matula. RLF** acquires the stable color sets $V_1 ... V_k$, which are sets of vertices for which it holds that for any two vertices in V_i there is no edge between them, by constructing them sequentially. Each V_i is obtained by repeating a 2 step procedure a number of times. The best one found so far is kept. In the first phase a greedy construction of a stable set I' is done, which leaves R remaining vertices. From the remaining vertices R a second stable set I'' is determined with a exhaustive search. $I' \cup I''$ is then a candidate as stable set for V_i.

For the combined procedure of RLF** and simulated annealing the RLF** is applied on the graph until at most a specified number of vertices is left uncolored. Then the simulated annealing procedure is applied on the resulting color sets. They found that combining RLF** with simulated annealing resulted in better solutions compared to applying just one of the algorithms, especially on larger graphs.

Also in 1987 A. Hertz and D. Werra published [16] their work on a tabu search algorithm modified for graph coloring named Tabucol. The local search strategy is equivalent to the SA-based algorithm of Chams et al. as they use the same solution space, neighborhood and objective function. Tabucol keeps track of recently visited configurations which enables it to avoid these for a certain number of iterations. This gives Tabucol a performance edge over SA as this enables Tabucol to escape from local optima, whereas SA would need to accept a successive number of worse moves in order to escape a local minimum. Later on, we will look into Tabucol in more detail. Hertz and Werra additionally propose combining Tabucol with a pre-processing step for large graphs with more than 500 vertices. Like with the combined method of Chams et al. this pre-processing step extracts stable color sets and the second algorithm is used to color the remaining vertices. Whereas Chams et al. used a greedy algorithm for finding the stable sets, Hertz and Werra used a simplified version of Stabulus, which is a tabu search algorithm for the maximal stable set problem [10]. The method tries to make stable sets of the largest possible size p. If successful, a class of a certain size has been found, otherwise we tried again for $p - 1$. The method works as follows: given size p and vertex set n consisting of vertices not yet assigned to a color class, the vertices are divided in two sets S and $\overline{S}$, where $|S| = p$. Then vertices are swapped between S and $\overline{S}$. The vertices to be swapped are selected by picking the vertex with highest number of neighbors within S from S and the vertex with lowest number of vertices within S from $\overline{S}$ which are not tabu. Vertices that have been swapped cannot be swapped again for a certain number of iterations. The objective function is to minimize the number of edges between vertices in S. If the objective function is 0 we have found a color class of size p. This method is repeated until the remaining nodes (not assigned to a color class) is below the specified threshold.

Johnson et al. compared three versions of SA algorithms in 1991 that employ different strategies [17]. The penalty function strategy uses a dynamic number of colors and tries to minimize the number of conflicts and the number of colors at the same time. The neighborhood used contains all possible configurations, both legal and not legal, consisting of $1 \leq k \leq |V|$ color classes. The objective function the algorithm tries to minimize is the following: $cost(C) = -\sum_{i=1}^{k} |C_i|^2 + \sum_{i=1}^{k} |C_i| * |E_i|$, where E_i, $1 \leq i \leq k$ is the set of edges with begin and endpoint in the color class i (i.e. the number of conflicts). Local optima of the objective function correspond to legal configurations, because if there were conflicts left a conflicting vertex could be moved to a new color class which would improve the value of the objective function (as conflicts weight more heavily on the cost function than the number of color classes).

The Kempe chain strategy they tested considers only legal solutions as possible moves and also uses a dynamic number of colors. As this strategy only considers legal configurations the cost function is simpler than the one of the previous strategy: $cost(C) = -\sum_{i=1}^{k} |C_i|^2$. On the other hand ensuring that moves preserve legality of the coloring requires a more complex sort of move involving Kempe chains interchanges. In a graph G with color classes V_i and V_j in a solution c, a Kempe chain is a connected component in a subgraph induced by $V_i \cup V_j$. A connected component is a subgraph in which any two vertices are connected by a path, and to which no vertices of the rest of the graph may be added without violating the connectedness (i.e. it is maximal). Consider a triplet (v,i,j) where $V_i \neq V_j$, $v \in V_i$ and the subgraph induced by $V_i \cup V_j$ is not connected. When such a triple has been found, a Kempe chain interchange for this triplet consist of replacing V_i by $(V_i - K) \cup (K - V_i)$ and V_j by $(V_j - K) \cup (K - V_j)$, where K is the Kempe chain for V_i and V_j containing v.

The last strategy they tested is the penalty strategy with a fixed number of colors as used by Chams et al for Tabucol. As with the penalty function, solutions are all possible configurations, but with the specified number of color classes, which may be empty. The objective function that should be minimized is simply the number of edges with both endpoints in the same color class. Moves consist of moving a single vertex to another color class. As only nodes with conflicts are considered, the number of possible moves decreases as the number of conflicts decreases.

The tests they performed on random graphs did not show a distinct advantage of using a specific strategy.

Also in 1991, L. Davis published his work on a genetic algorithm (GA) for the graph coloring problem [7]. GA's are optimization procedures in which populations of individual solutions evolve in a manner similar to evolution and natural selection. This means these procedures have some means to recombine solutions into new solutions and select a part of the population which is deemed worse than the rest of the population and are to be discarded.

The algorithm Davis proposed encodes solutions as permutations of the vertices. This is known as order-based encoding. Then to evaluate a solution, a greedy algorithm is applied which walks through the graph in the order specified by the permutation and assigns an available color to the vertices. To generate offspring, solutions are combined by doing a order-based uniform crossover between two parent solutions. This is done by selecting assignments from the first parent with 50% chance. Remaining assignments are taken from the second parent. Next, two vertices in the offspring are randomly selected and all vertices between these vertices are scrambled (according to vertex numbering). To give good performing configurations a larger chance of being selected as the parent of offspring, the population has a limited size. If this limit is exceeded the worst configuration (with the highest number of conflicts) is dropped from the population.

While the algorithm improves on greedy algorithms, as compared to other algorithms this GA performs not very good.

In 1995 Costa et al. were the first to publish results of experiments with genetic local search (GLS) [6]. The difference between GA and GLS is that whereas Davis' GA only mutates offspring, Costa et al.'s GLS also tries to improve the offspring by means of a local search operator. In order to facilitate this they replaced the order-based encoding with a string-based encoding, which is also used by local search algorithms like Tabucol. This encoding stores the color class of each vertex directly.

The crossover operator used by Costa et al. creates offspring from two parents by deciding for each vertex from which parent the color should be used. From which parent the color is used is decided by determining which of both colors yields the lowest penalty from a penalty function. The penalty function measures the "closeness" of conflicts to the current vertex, where closer conflicts weigh more heavily. Mutations can be performed by randomly swapping the color class of each vertex with a low probability. Additionally, to optimize a newly created solution, Costa et al. used vertex descent. Vertex descent works by sequentially performing moves which reduce the number of conflicts, until no moves are left which reduce the number of conflicts. The moves used consist of moving a single vertex to a different color class. The evolutionary descent method which uses these components will repeat he following steps while the stopping conditions, which are finding a valid solution or reaching the specified maximum number of iterations, are not met.

1. To generate new solutions, the crossover operator is applied to solutions of the population.
2. Mutate all newly generated solutions.
3. Optimize the new solutions with vertex descent.
4. Replace the worst solution in the population with the best newly generated solution.

Costa et al. tested this evolutionary descent method (EDM) on random graphs and found that it performed better than Tabucol.

In 1996 Fleurent and Ferland also experimented with GLS [9], but they used tabu search as LS operator instead of a descent method as used by Costa et al. For generating offspring multiple crossover operators were used: 1- and 2-point, uniform crossover and graph-adapted crossover. 1-point crossover picks a vertex randomly and all vertex colors of vertices up to this vertex number are inherited from the first parent. The other vertex colors are inherited from the second parent. 2-point crossover is similar to 1-point crossover but picks 2 points randomly and inherits vertex colors from vertices with numbers between these two values from the first parent, and outside this range from the other parent. With Uniform crossover, for every vertex there is 50% chance it will be assigned the color class of the color class of parent 1 and 2, respectively. The graph-adapted crossover operator is a new operator which is based on conflicting vertices: vertices are colored according to the parent which has no conflicts for that vertex. If both parents have conflicts for that vertex the color used least in neighboring vertices in both parents is used.

In the experiments Fleurent and Ferland conducted they found that their GLS algorithm outperforms Tabucol. However, for large graphs with 500 to 1000 vertices their GLS algorithm was not able to find a legal solution in a reasonable time (48 hours). This caused them to combine their algorithm with a greedy algorithm to extract stable color sets beforehand, as has been done by Hertz and Werra before [16]. This resulted in a significant improvement, at the cost of additional computational complexity. Performance also differs between the recombination operators with respect to the speed they help the search to converge to a legal solution. The 1-point and 2-point crossover operators are outperformed by uniform crossover, which in turn performs less than the graph-adapted crossover.

In 1993 Morgenstern came up with a new LS strategy [23] which considers solutions containing legal partial colorings. Then the objective is to minimize the sum of degrees of nodes not assigned to a color class. Moves consist of assigning a color to an uncolored vertex and resolve conflicts by uncoloring neighboring vertices which belong to the same color class. Morgenstern used SA as LS method and tested several variants of the algorithm. The best performing version uses a population initialized by the XRLF procedure of Johnson et al., which extract stable sets. This allowed them to achieve better results on large graphs than known at that time, at the expense of a larger complexity.

At the end of the 1990s new GLS algorithms were proposed. The first by R. Dorne and J.K. Hao [8] and the second by J.K. Hao and P. Galinier [15]. Both algorithms use Tabucol as LS method. In contrast to many earlier good performing algorithms these two algorithms do not use preprocessing to extract stable color sets, but they outperform them nonetheless. The reason that these algorithms perform so well is that they use a new crossover operator which recombines color classes instead of specific color assignments. This is good as symmetric solutions, whose color classes contain the same nodes, are correctly combined and as such more color classes are inherited by the offspring.

Dorne and Hao use the union independent set crossover operator, which tries to unify pairs of independent color sets (I_{p_1}, I_{p_2}) (sets without conflicts) from parent 1 and 2 respectively. This is done by performing the following for each parent: let $I_{p_1,c}$ be the largest conflict-free subset of vertices (an independent set) having color c in parent 1. Then we search for the largest conflict-free subset in parent 2 such that the number of vertices these subsets have in common is maximal: $\forall c' \in [1...k], |I_{p_1,c} \cap I_{p_2,c'}|$ is maximal. This is repeated for every color. The resulting sets of unions are used to generate child e_1 and child e_2 respectively, by assigning the color of $I_{p_1,c}$ to the vertices in $I_{p_1,c} \cap I_{p_2,c'}$ for e_1. Vertices not included in the set of unions keep their original color. For e_2 the procedure is same, only p_1 is replaced with p_2.

Hao's and Galinier's Hybrid Coloring Algorithm (HCA) uses the Greedy Partition Crossover (GPX) operator to generate new offspring from two parents s_1 and s_2. GPX then repeats the following for each color c: choose the color class with largest cardinality from s_1 if c is even or s_2 otherwise. Assign the vertices of this color class to the color class V_l of the offspring. The vertices that were just assigned are then removed from the color class of both s_1 and s_2. When vertices have been assigned to every color class of the offspring, vertices missing from the configuration will be randomly assigned to color classes.

These GLS algorithms are able to find the best known solutions for many commonly used graphs and find better solutions for some large graphs.

A optimization technique called Variable Neighborhood Search (VNS) was proposed by Hansen and Mladenović [14] in 2001 which uses a combination of several neighborhoods during searching. The motivation for this is that many heuristic algorithms for combinatorial optimization problems had focused on avoidance of being trapped in local optima while the use of alternative neighborhoods remained largely unexplored. In contrast to other local search algorithms VNS does not follow a trajectory, but instead explores increasingly distant neighborhoods. The move to another solution from a current one is only made if a improvement is made. This should preserve good characteristics of the current solution. Additionally, local searches are performed to reach the local optimum of the neighboring solutions.

Avanthay et al. applied VNS to the graph coloring problem using Tabucol as LS operator [2]. This proved to be more efficient than using Tabucol standalone.

In 1995 Rochat and Taillard proposed the meta-heuristic Adaptive Memory Algorithm (AMA), which instead of keeping a population and using a crossover operator to recombine solutions like GLS does, stores portions of solutions and recombines them into new solutions [27]. They used this algorithm to solve the traveling salesman problem. After creating the initial set of partial solutions a generation consists of generating offspring by combining partial solutions, performing a local search on the offspring and, lastly, use parts of the resulting solution to update the set of partial solutions.

Galinier et al. adapted the AMA method for the graph coloring problem under the name Amacol [11]. As partial solutions stable color sets are used, which are combined into a new solution. Then Tabucol is applied to the new solution to color remaining vertices. If the stopping condition is not met (i.e. no valid solution has been found) the stable color sets of the new solution are inserted into the set of partial solutions replacing randomly selected elements. The results found by Amacol are comparable with those found by other GLS algorithms.

In 2008 I. Blöchliger and N. Zufferey [3] proposed a variant of Tabucol which has a reactive component to adjust the length of time certain moves are excluded based on the fluctuations of the objective function. This is based on the idea that small fluctuations during a long time indicate that the search process has been stuck in a local optimum and the algorithm should take measures to escape [3]. This is done by checking periodically whether the difference between highest and lowest number of conflicts in the last time frame is below a certain threshold. If so, the number of iterations moves are forbidden after applying them is increased. This will increase chances of climbing out of a local optimum. In contrast to Hertz and Werra's Tabocol which starts from an non-legal solution, Blöchliger and N. Zufferey use the partial k-colorings approach like Morgenstern [23] did before. The results are competitive with other algorithms while being relatively simple and efficient. The reactive component can also be used to improve the performance of Hertz and Werra's Tabucol.

An analysis of the search space of a local search on a graph coloring problem was published by Porumbel, Hao and Kuntz in 2009 [25]. By measuring the distance between solutions they were able to record the spatial distribution of the best solutions visited by a single run of a local search algorithm. They found that solutions of high quality were not distributed evenly over the search space, but were clustered together in spheres of limited size. The size of the sphere does not depend on the type of the graph, but mainly on the number of vertices in the graph. Depending on the graph, the diameter of the spheres varies from 7% to 10% of $|V|$.

Porumbel, Hao and Kuntz developed a search algorithm to exploit this spatial distribution. Their TS-Div algorithm, also referred to as Position Guided Tabu Search (PGTS), tries to guide the search process by leading the search process away from regions in the search space which were already visited. The idea is that this will ensure the search algorithm will explore the entire search space, in other words ensure diversification. It does this by keeping an archive of spheres (regions in the search space) and after making a move, checking whether the new configuration is still contained within the last visited sphere. If not, the algorithm checks whether this new configuration is contained in a sphere visited before. If this is not the case a new sphere is created and added to the archive. If the configuration is part of an already visited sphere the current region in the search space should be avoided as it was visited before. The algorithm tries to accomplish this by increasing the number of iterations for which moves are made tabu. In this way the search process is guided out of the sphere, as making moves which decrease the distance towards the border of the sphere will not be allowed to be reversed for longer time thus giving the search process a larger chance to move on to another region.

A problem with the TS-Div algorithm is that it might be too aggressive with its push for diversification. A sphere which contains an optimal solution might be visited by the search process while missing the optimal solution. As the search process tries to avoid visiting spheres again, the optimal solution might never be found. To remedy this Porumbel, Hao and Kuntz came up with the TS-Int algorithm, which explores a given configuration intensively. Given a configuration C_s, TS-Int launches a number of tabu searches which explore the proximity of C_s. The resulting configurations are kept in a queue, ordered according to their quality. Then TS-Int picks the head of the queue and launches TS processes on this configuration. The resulting configurations are inserted into the queue. This is then repeated.

TS-Int can be applied to the best configurations found with TS-Div. This combination is competitive to the best algorithms to date.

In 2010, a new evolutionary algorithm known as MACOL was published by Zhipeng Lü and Jin-Kao Hao which uses a multi-parent crossover operator [21]. This crossover operator, which is an extension to GPX, can be used to create offspring from two or more parents, instead of exactly two. AMPaX builds the color classes for the offspring by selecting the largest color class out of the parent population. After adding this class to the offspring, all vertices contained in this class are removed from all parent color classes. This is repeated until the required number of color classes has been reached. Remaining vertices are added randomly to a color class, just as with GPX. As the color classes can come from any of the selected parents instead of just two, the offspring potentially inherits larger color classes. As this results in fewer unassigned vertices, the offspring has a higher probability of becoming a legal configuration.

To ensure that the population of configurations remains diverse, AMPaX decides whether to add a configuration to the population on the basis of its distance to members of the population and its quality. This is done by temporarily adding the candidate to the population and calculating a score for each member. The score of a configuration consists of the number of conflicts and the distance to the other configurations, where distance is the minimal number of 1-moves required to get from one configuration to the other. Then the population member with the highest score (i.e. least desirable) is removed from the population.

MACOL obtains very competitive results on many of the benchmark graphs.

Daniel Porumbel, Jin-Kao Hao and Pascale Kuntz proposed another evolutionary algorithm in 2010 [26], which like Zhipeng Lü's and Jin-Kao Hao's MACOL tries to maintain population diversity and generates offspring using a multi-parent crossover operator. The Well-Informed Partition Crossover operator (WIPX-crossover) of their Evo-Div algorithm uses randomly selected parents to generate offspring and then selects color classes from these parents according to a scoring function. The score of a color set is equal to:

$$conflicts - \frac{1}{|v|}(classSize + \frac{degreeCls}{|E| * |V|}) \tag{2.1}$$

In equation 2.1 $classSize$ is the cardinality of the color set in question and $degreeCls$ the sum of the degrees of the class vertices. The best color classes have low scores on this function, as they have a lower number of conflicts and a large number of nodes. A larger sum of degrees also leads to a better score as vertices with a high degree are more difficult to color so choosing a color class with a large sum of degrees leaves easier vertices uncolored.

Evo-Div's population diversity strategy is to ensure that the minimal distance between any two configurations in the population is kept above threshold R and that the average distance between configurations is maximized. As with AMPax, the distance between two configurations is the minimal number of vertices that have to be moved between color classes to make the two configurations equal. The threshold R is chosen according to clustering of related solutions as found by Porumbel, Hao and Kuntz in 2009 [25]. By choosing $R = |V|$ the minimal distance between configurations will ensure that the configurations in the population will not be closely related. If the average distance between individuals in the population is less then $2R$ and all individuals have the same fitness value, which might be due to the structure of the search space like multiple plateaus within a deep well, a dispersion mechanism is triggered. This dispersion mechanism mutates the offspring to increase their distance to individuals of the population.

Evo-Div performs very well on benchmark graphs, reproducing the best known solutions on almost all graphs and finding a better than previously known solution for one particular graph.

2.2 Local search strategies

A LS algorithm consist of three parts: the search space to be explored, the evaluation function to be minimized and the neighborhood function. Following Galinier and Hertz [12] the LS strategies for the graph coloring problem can be divided into four categories.

2.2.1 Legal strategy

This strategy uses a search space that contains all legal colorings and the objective is to find a solution that uses as few colors as possible. As simply moving vertices to another color class will rarely lead to a legal coloring a more sophisticated kind of moves will have to be employed. The Kempe chain interchange of Morgenstern and Shapiro [22] is a better neighborhood function. In a graph G with color classes V_i and V_j in a solution c, a Kempe chain is a connected component in a subgraph induced by $V_i \cup V_j$. A connected component is a subgraph in which any two vertices are connected by a path, and to which no vertices of the rest of the graph may be added without violating the connectedness (i.e. it is maximal). Consider a triplet (v,i,j) where $V_i \neq V_j$, $v \in V_i$ and the subgraph induced by $V_i \cup V_j$ is not connected. When such a triple has been found, a Kempe chain interchange for this triplet consist of replacing V_i by $(V_i - K) \cup (K - V_i)$ and V_j by $(V_j - K) \cup (K - V_j)$, where K is the Kempe chain for V_i and V_j containing v. A Kempe chain interchange is only performed in case the induced subgraph is not connected, as it will only swap color classes otherwise.

As evaluation function the number of color classes could be used. However, as moves that reduce the number of colors are rare, this would not be so useful, therefore Morgenstern and Shapiro proposed a different evaluation function: $-\sum_{i=1}^{k} |V_i|^2$. When this function is minimized the size of the smaller color classes will decrease to the point where they can be removed.

2.2.2 K-fixed partial legal strategy

Like the legal strategy, the search space contains only legal colorings. But now the number of colors is fixed and solutions can be partial. A solution c in the search space can be represented by a partition $(V_1, ...V_k, V_k + 1)$ such that $V_1, ...V_k$ are the legally colored vertices and $V_k + 1$ the set of uncolored vertices.

Like with the legal strategy Kempe chain interchanges can be performed. But this search strategy allows for more simple moves called i-swaps as proposed by Morgenstern [23]. An i-swap involves moving a vertex v from the set of uncolored vertices to V_i. Then all vertices adjacent to v that belong to V_i as well are moved to $V_k + 1$. As an objective function, minimizing the number of vertices in $V_k + 1$ can be used. Another possibility is using the function $\sum_{v \in V_k+1} d(v)$, where $d(v)$ is the number of vertices adjacent to v.

2.2.3 K-fixed penalty strategy

This strategy, which uses a fixed number of color classes k, has a search space which consist of all k-colored solutions, whether they are legal or not [16]. The neighborhood of a configuration contains all configuration that can be reached by changing the color of a single vertex. The objective function is simply the minimization of the number of conflicting edges, that is to minimize $\sum_{i=1}^{k} |E_i|$, where E_i denotes edges between two vertices in the same color class.

2.2.4 Penalty strategy

In this strategy the number of colors is not fixed and as with the k-fixed penalty strategy the search space contains both valid and invalid solutions. The objective function requires the minimization of both the number of conflicts as well as the number of colors. This is done by minimizing $\sum_{i=1}^{k} 2\,|V_i|\,|E_i| - \sum_{i=1}^{k} |V_i|^2$. The second term of the objective function decreases the smaller color classes and thereby decreases the number of colors. Johnson et al. proved that local optima reached with this objective function correspond to legal colorings [17].

2.3 Tabucol

Tabucol is a local search algorithm for finding a solution to the $k{-}GCP$ problem [16] using the k-fixed penalty strategy. It is a modification of tabu search as proposed by Glover [13]. An important change Hertz and Werra made is what is made tabu. Instead of making a whole configuration tabu, only single moves are made tabu. Neighbors of configurations will be considered tabu if the move that leads to them has been marked tabu. The outline of the algorithm is shown in Algorithm 2.3.1. The version of tabu search for graph coloring presented here differs slightly from Hertz and Werra's Tabucol and was published by D.C. Porumbel, J.-K. Hao and P. Kuntz [24], as this version makes moves tabu for a variable number number of iterations, instead of making all moves tabu for the same number of iterations. Tabocol was chosen for this thesis project because it is powerful, yet the algorithm is relatively simple compared to other local search algorithms.

Tabucol starts by generating a random coloring on the given graph G using k color classes. This will (almost certainly) not be a legal coloring. The objective function f_c which Tabucol is trying to minimize is equal to the number of conflicts in a configuration: $f_c = \sum_{n=1}^{k} |\forall (i,j) \in E : c(i) = c(j)|$. If a valid k-coloring C has been found $f_c(C) = 0$.

Tabucol will then start to explore the search space Ω, which consist of all possible colorings of G: $|V|^k$. Neighbors C' of a configuration $C \in \Omega$ are obtained

Algorithm 2.3.1: TABU SEARCH(G, k)

```
Initialize random configuration C
while $conflicts \neq 0$ and not reached iteration or time limit
  do {
    comment: Find non-tabu neighbor with largest improvement for the
    objective function. Ties are broken randomly.
    $C' \leftarrow findLargestImprovementMove(N(C))$
    comment: Make the move
    $C \leftarrow C'$
    if $f_c(C)$ unchanged
      then $increment(m)$
      else $m \leftarrow 0$
    $T_l \leftarrow \alpha * f_c(C) + Random[0, A] + m/mmax$
    $makeTabu(move, T_l)$
    if $f_c(C) < f_c(C_{best})$
      then $C_{best} \leftarrow C$
return $(f_c(C_{best}))$
```

by changing the color of a single vertex.

After the initial coloring has been generated Tabucol will start iterating until a stopping condition is met, which is either finding a legal solution or reaching a time or iteration limit. Then the lowest number of conflicts that has been found is returned. As can be seen, each iteration consist of picking a next move, applying it and making the move tabu for a certain number of iterations.

Selecting a next move is done as following. From all possible moves the ones with the largest improvement to f_c are selected. Moves that are tabu are excluded, unless they improve the best known result for the graph as a whole, or improve the best known solution of a node. These exceptions are not part of the original Tabocol, but have been added to ensure new better regions in the search space are not overlooked. If the resulting set of best moves contains more than one move, one is picked at random.

After a move is chosen, the move is applied, in other words we move to the neighboring configuration. The move that was just applied should not be applied for a certain number of iterations. The exact number of iterations is determined by the tabu tenure T_l. T_l depends on the objective function and the number of iterations the objective function has remained unchanged. More precisely:

$$T_l = \alpha * f_c(C) + random[0, A] + \lfloor \frac{m}{mmax} \rfloor \tag{2.2}$$

In equation 3.1, α is a parameter with a non-negative value to control the impact of the number of conflicts on the objective function, $random[0, A]$ a random value between 0 and A and $\lfloor \frac{m}{mmax} \rfloor$ is a reactive component to help the algorithm to move on to a new part of the search space when it has been cycling on a search plateau for a while. How soon this occurs is determined by the size of mmax. The purpose of the random number between 0 and A was not mentioned by D.C. Porumbel, J.-K. Hao and P. Kuntz, even though it is not part of the original Tabocol. But whereas Tabucol had a tabu list of fixed length, which functions as a first-in first-out queue, meaning that moves will be guaranteed tabu for the length of the queue, the version of tabu search presented has no such guarantee. It is likely that this term is a compensation for this, but we will look into this later.

2.4 Position-guided tabu search

In addition to avoiding configurations we have visited so far, it could also be beneficial to avoid regions in the search space S(C) we have investigated thoroughly already. This is what position guided search as proposed by D.C. Porumbel, J.-K. Hao and P. Kuntz [24] does. It accomplishes this by keeping a archive of configurations visited so far. However, this archive does not contain every configuration that was visited as this would make the archive too large to be of any practical use. Instead a configuration is only added to the archive if it represents a new region. So essentially the archive is a collection of regions in the search space. Algorithm 2.4.1 gives an overview of the PGTS algorithm.

Porumbel, Hao and Kuntz defined a region as a sphere centered at a configuration C with radius R. Given the distance function $d : \Omega \times \Omega \rightarrow \mathbb{N}$, a configuration $C \in \Omega$ and radius $R \in \mathbb{N}$, the R-Sphere S(C) with its center (also called pivot point) at C contains the set of configurations $C' \in \Omega$ for which it holds that $d(C, C') \leq R$.

The distance used is the minimal amount of moves required to make the configurations equal. More precisely, for a move from C to C' this means there should be a path $C_0, C_1, C_2, ..., C_n \in \Omega$ such that $C_0 = C, C_n = C'$ and $C_{i+1} \in N(C)$ for all $i \in [0...n-1]$.

After the vertices of the graph have been assigned a color randomly to create the initial configuration, this configuration is made the pivot point. The pivot point serves as center of a sphere in the search space, which has the lowest number of conflicts of the known configurations in the sphere.

Just as with Tabucol, the best non-tabu neighbor is selected and the configuration is moved to this neighbor. Then there is an added step which is only executed if the distance between the current configuration and the pivot point is larger than radius R. If the distance is larger than R the search process has left the last sphere. Therefore the distance to the new configuration is compared to spheres in the archive. If the new configuration falls within a recorded sphere, in other words if this region has been visited before, T_c is incremented. T_c is an additional component for the tabu tenure, which purpose is to give moves leading to visited regions a higher number of tabu iteration. If a lot of already visited regions are visited again this will lead to more active exploration. If the current configuration is not contained in an already visited region, T_c is set to 0 and the current configuration is added to the archive.

Then reversing the move (giving the vertex the color it had before the move just made) just performed is made tabu for $T_l + T_c$ iterations, where T_l is calculated just as it was in the case of the Tabucol. Lastly if the current configuration has fewer conflicts than the pivot configuration the sphere is 're-centered' by making the current configuration the pivot point.

As each iteration uses at least one distance calculation, being able to calcu-

Algorithm 2.4.1: PGTS(G, k)

```
Initialize random configuration C
$C_p \leftarrow C$
$T_c \leftarrow 0$
while $conflicts \neq 0$ and not reached iteration or time limit
  do
    comment: Find best non-tabu neighbor
    $C' \leftarrow findBestMove(N(C))$
    comment: Make the move
    $C \leftarrow C'$
    if $d(C, C_p) > R$
      then
        $C_p \leftarrow C$
        if $alreadyVisited(C_p)$
          then $increment(T_c)$
          else
            $T_c \leftarrow 0$
            $record(C_p)$
    if $f_c(C)$ unchanged
      then $increment(m)$
      else $m \leftarrow 0$
    $T_l \leftarrow \alpha * f_c(C) + Random[0, A] + m/mmax$
    $makeTabu(move, T_l + T_c)$
    if $f_c(C) < f_c(C_{best})$
      then $C_{best} \leftarrow C$
    if $f_c(C) < f_c(C_p)$
      then
        Replace $C_p$ in archive with $C$
        $C_p \leftarrow C$
return $(f_c(C_{best}))$
```

late this efficiently is important. To make two configurations equal there exists a color relabeling σ such that $C_a^i = C_b^{\sigma(a)}$, where $1 \leq i \leq k$. Then the coloring distance can be seen as a partition distance: the minimal number of vertices that have to switch color class.

To calculate this distance the Hungarian algorithm can be employed, which was proposed by Harold W. Kuhn [19]. The Hungarian algorithm is a optimization algorithm which solves the assignment problem in polynomial time ($O(n^4)$). However, in practice this is still a computational intensive function. In fact we are not interested in the precise distance at all, as we just want to know whether the distance between two configurations is larger or smaller than R.

To determine this, the following Las Vegas method can be used. It works by determining the maximal and minimal amount of similarity. This is done by iterating through all $x \in V$ and incrementing a matching counter between color class $C_a(x)$ of C_a and $C_b(x)$ of C_b. The result is a $k * k$ matrix, which contains for every color class i from C_a and j from C_b the number of vertices which are contained in both color classes if color class i was assigned to color class j. For all color classes C_a^i there should be a best match (highest matching number) denoted by $C_a^{\sigma(i)}$. The similarity will be at most $\sum_{i=1}^{k} \left| C_a^i \cap C_b^{\sigma(i)} \right|$. Knowing the maximal similarity $s(C_a, C_b)$ is useful if we want to find out whether the distance between two configurations is larger than R, as the distance between two configurations is at least $|V| - s(C_a, C_b)$. If this is larger than R we have found the answer we sought. If on the other hand we want to know whether the distance is smaller than R, we can use the similarity if we keep the same color classes between the configurations. If $|V| - s(C_a, C_b) < R$ it is certain that the distance is smaller than R. If the Last Vegas method did not yield an answer the Hungarian algorithm will have to be used. But in practice this occurs in less than 10% of the distance calculations. The choice of parameter R is also important as it controls the size of the regions. If $R = 0$, all visited configurations would be saved to the archive as separate regions (as distance between them is always larger than 0). On the other hand if $R = |V|$ all configurations would fall into the same region.

As mentioned in the historical overview, to find a value for R, Porumbel, Hao and Kuntz did a statistical analysis of a tabu search run. The result of this analysis was that most distances between configurations were either small ($\approx$ 5% of $|V|$) or large ($\approx$ 40% of $|V|$). As the configurations which are close together are very similar and the configurations which are further apart are not very similar, it is logical to pick a size for the spheres somewhere in between. This led them to pick 10% of $|V|$ as R.

Chapter 3

Experimentation

3.1 Implementation

PGTS was implemented using the Java programming language. Implementation of PGTS was relatively straightforward. The main PGTS function contains a switch to turn position-guidance on or off to allow experimentation with the basic Tabucol algorithm.

Tabucol uses a $|V| \times k$ array to keep track of the change in penalty for each move. This is generated at the beginning and partially updated when moves are made. An array of the same dimensions is used to keep track of tabu status of moves. This allows Tabucol to efficiently determine which moves are the best choice.

3.2 Goals

Tabu search for graph coloring as used by D.C. Porumbel, J.-K. Hao and P. Kuntz [24] has three parameters, which all influence the tabu tenure T_l, i.e. the number of iterations moves are not to be chosen. T_l is calculated as following:

$$T_l = \alpha * f_c(C) + random[0, A] + \lfloor \frac{m}{mmax} \rfloor \tag{3.1}$$

Firstly there is α which determines the weight of the number of conflicts on the tabu tenure. The parameter A, which is used for drawing a random value between 0 and A is added to the tabu tenure. And lastly $mmax$, which only comes into play when the number of conflicts has been unchanged for at least the last $mmax$ iterations, in which case this number of iterations is divided by $mmax$ and added to the tabu tenure. Porumbel, Hao and Kuntz recommended these parameters to be set to 0.6, 10 and 1000, for α, A and $mmax$ respectively. However, no motivation for choosing these specific parameter values is given.

Having multiple parameters with seemingly arbitrary value is undesirable as it is unclear as to why these values were chosen. The research goal will be to determine the impact of these parameter values. Following these results it might be possible to do without a number of parameters. The first step will be finding out the sensitivity of the search to variation of the parameters. This will tell us how robust these parameter values are. The next step will be to find out whether all three parameters are necessary for achieving good results.

3.3 Benchmark graphs

Experiments are conducted on graphs of the DIMACS Challenge Benchmark [1]. The graphs differ in the way they were generated and only for a subset of the graphs the minimal number of colors required for legal coloring is known (χ, the chromatic number). The graphs of the DIMACS Challenge Benchmark can be divided into four classes:

1. Classical random graphs with unknown chromatic number (*dsjc500.5* and *dsjc1000.1*, where the first number is $|V|$ and the second number denote the graph density, i.e. the chance for any two nodes to be connected).
2. the "Leighton graph" (*le450_25c*) with $\chi = 25$ which contains at least one clique of size χ.
3. a "flat" graph (*flat300_28_0* with $\chi = 28$) which was generated by partitioning the vertices into χ color classes and allowing only edges between vertices of different color classes.
4. *r1000.1c*, a geometric graph generated by picking points uniform at random from a square and by making edges between all pairs of vertices within a certain distance.

For the experiments with a graph the number of colors k is chosen such as to make it difficult for the search algorithm to find a legal configuration within the limited number of iterations allowed.

3.4 Robustness of parameters

To find out how sensitive the performance of Tabucol is to variation in its parameters, we will run experiments were one parameter is varied while the other two parameters are fixed to the values as specified by Porumbel, Hao and Kuntz. These values are as following: $\alpha = 0.6$, $A = 10$ and $mmax = 1000$. For every graph these experiments consisted of 32 independent instances of the search process, from here on referred to as runs. This should ensure that any differences found are statistically significant. Every run is limited to 40 million iterations, which should give the search algorithm enough time to find a solution. To test whether the results for different parameters are significantly different the Kruskall-Wallis test is used, which is a non-parametric method for testing equality of population medians among groups. If the resulting p-value is smaller than the significance level of .05, we reject the null hypothesis that the medians of the groups are equal and consider the performance differences to be significant.

3.4.1 Varying α

First we will see how performance (i.e. the resulting number of conflicts) is affected when varying the parameter α. D.C. Porumbel, J.-K. Hao and P. Kuntz [24] used $\alpha = 0.6$, so we are comparing the results of using this value for α with the results obtained when setting $\alpha = \{0.0, 0.3, 0, 9, 1.2\}$. These values were chosen because they are twice as large or small as the value used

Table 3.1: P-values of performance differences between Tabucol parameter values for multiple graphs. k specifies the number of colors allowed. If the p-value is smaller than the significance level of .05 we reject the null hypothesis that the medians of the groups are equal and consider the performance differences to be significant

Graph	k	α	A	$mmax$
dsjc500.5	49	0.000	0.000	0.529
dsjc1000.1	20	0.000	0.000	0.935
le450_25c	25	0.000	0.000	0.955
flat300_28_0	29	0.000	0.000	0.836
r1000.1c	98	0.000	0.000	0.279

by Porumbell, Hao and Kuntz, and as such should yield a different result if the algorithm is sensitive to its value. In figure 3.1 it can clearly be seen that parameter α has a very large influence on the performance of Tabucol on the graph djsc500.5. In figure 3.2 this influence can be seen as well. The effects are somewhat different, as for *dsjc500.5* the best performance is achieved for $\alpha = 1.2$, whereas $\alpha = 0.6$ gives the best performance for *flat300_28_0.* The graphs *le450_25c* and *r1000.1c* show similar results to *dsjc500.5.* The figures of these graphs have been included in appendix A.1. A small α affects performance very negatively, while for values of $\alpha \geq 0.6$ differences in performance are smaller. In table 3.1 an overview is given of the p-values of the differences in performance when using the different parameters. For parameter α, the p-values are much smaller than the significance level .05 for all the tested graphs, and thus the difference is very significant for all tested graphs.

3.4.2 Varying A

Parameter A, controlling the upper limit of the random value added to the tabu tenure is varied for graph *dsjc500.5* in figure 3.3. The same can be seen for graph *flat300_28_0* in figure 3.4. D.C. Porumbel, J.-K. Hao and P. Kuntz [24] used $A = 10$, and we compare it to $A = \{5, 20, 30\}$. For graph *dsjc500.5* the performance is best when using $A = 20$, while the best value for the graph *flat300_28_0* is harder to determine as $A = 10$ and $A = 20$ perform comparable. The impact of the parameter A appears to be not as quite as large as α, but the differences are still very significant as it can be seen in table 3.1 that the p-values are much smaller than the significance level .05 for all tested graphs. For figures of the graphs *dsjc1000.1*, *le450_25c* and *r1000.1c* see appendix A.2.

3.4.3 Varying $mmax$

Parameter $mmax$, which makes T_l larger when the number of conflicts does not change for $mmax$ number of iterations, is set to 1000 by D.C. Porumbel, J.-K. Hao and P. Kuntz [24]. Values used in comparison are $mmax = \{500, 1500, 3000, 5000\}$. Such large differences between values of $mmax$ should bring to light the possible sensitivity to this parameter.

Different values for parameter $mmax$ do not influence the performance of Tabucol greatly, as it can be seen in figure 3.5 for graph *dsjc500.5* and figure 3.6

Figure 3.1: The resulting number of conflicts after applying Tabucol with different values of parameter α on graph *dsjc500.5* with $k = 49$.

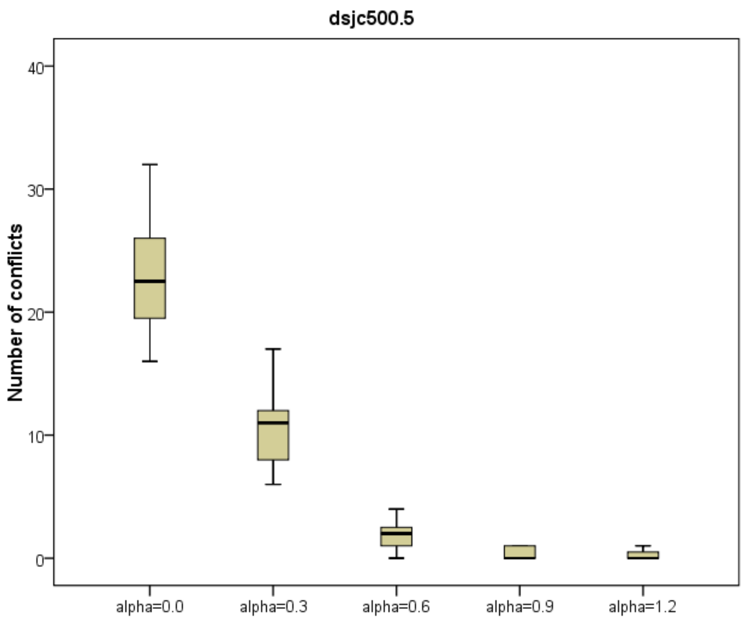


Figure 3.2: The resulting number of conflicts after applying Tabucol with different values of parameter α on graph *flat300_28_0* with $k = 29$.

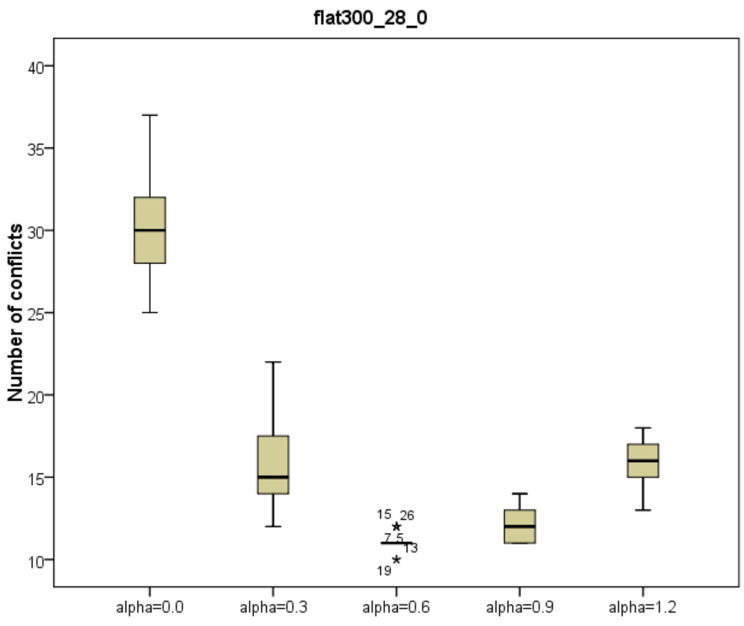


Figure 3.3: The resulting number of conflicts after applying Tabucol with different values of parameter A on graph *dsjc500.5* with $k = 49$.

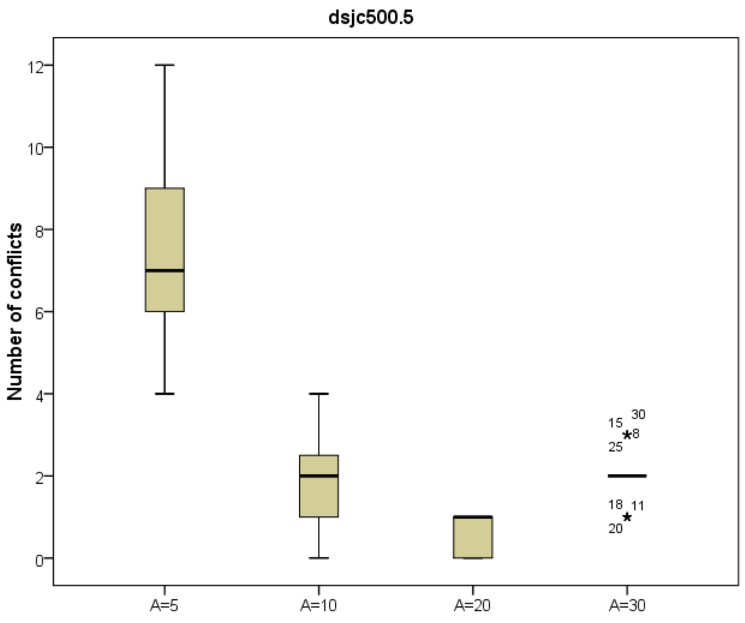


Figure 3.4: The resulting number of conflicts after applying Tabucol with different values of parameter A on graph *flat300_28_0* with $k = 29$.

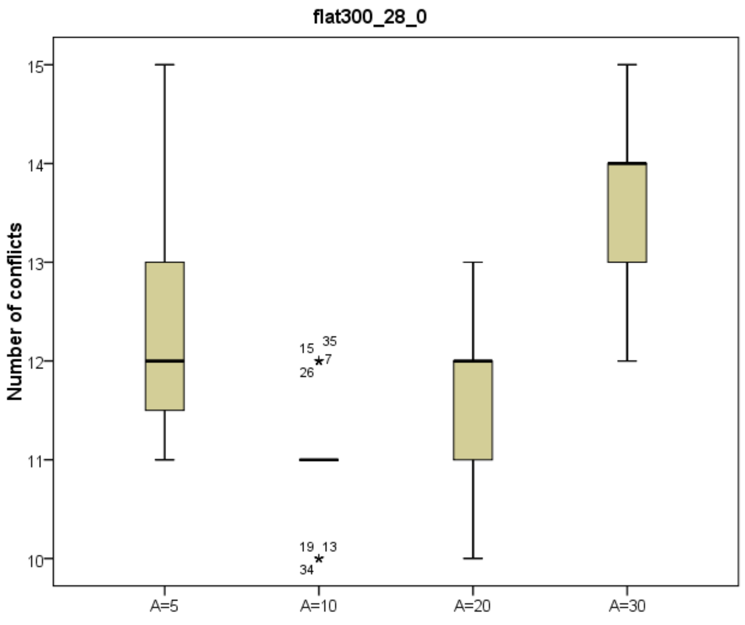
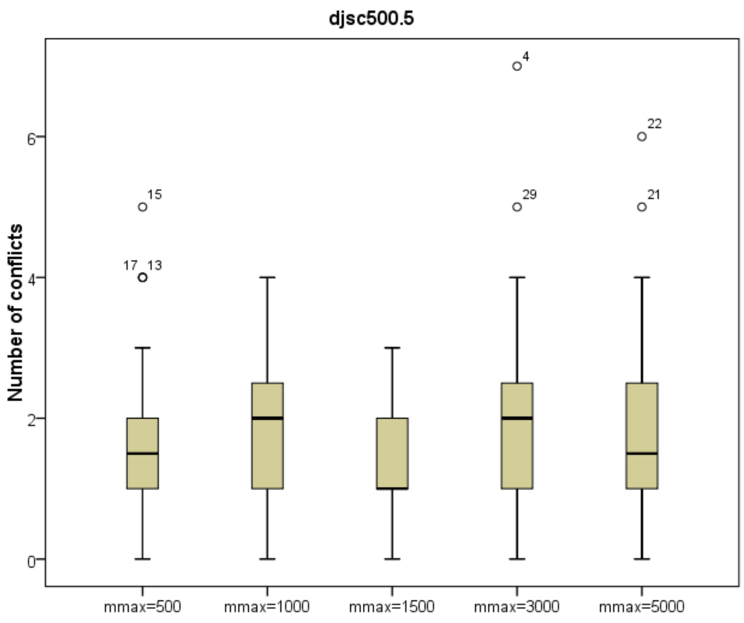


Figure 3.5: The resulting number of conflicts after applying Tabucol with different values of parameter *mmax* on graph *dsjc500.5* with $k = 49$.



for graph *flat300_28_0* that the number of conflicts found is very close together for all parameter values. The performance of the different parameters is not significantly different for any of the tested graphs (see table 3.1 again). Figures for the other graphs can be found in appendix A.3.

3.4.4 Hypothesis for performance differences

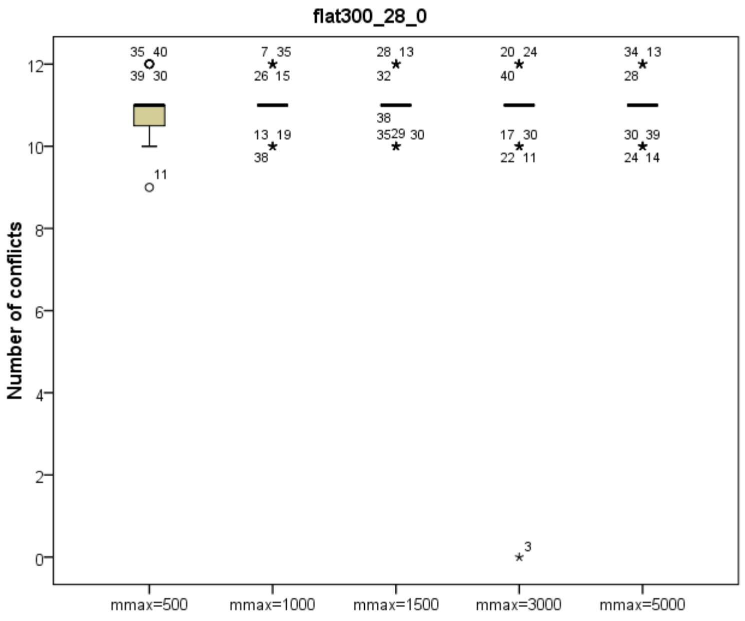
In the previous sections it has become clear that good values for parameters α and A are important to make Tabucol perform well. But why does performance suffer so dramatically when parameters α and A are given small values? My hypothesis is that small values for these parameters results in a small tabu tenure and thus moves are tabu only for a short number of iterations. This results in poor performance due to an inability of the search algorithm to escape from local optima. An effect of this should be that the search algorithm will visit the same configurations more often. In other words cycles occur. Whether this is the case is the subject of the next section.

3.5 Cycles

3.5.1 Detecting cycles

To test whether cycles are the cause of low performance of certain tabu parameters the algorithm was modified to detect whether a newly visited configuration was visited before. This is done by storing previously visited configurations and

Figure 3.6: The resulting number of conflicts after applying Tabucol with different values of parameter *mmax* on graph *flat300_28_0* with $k = 29$.



comparing them to the configuration just visited. However, this is a very time-consuming operation and has to be limited to be kept usable. This is why only the last 1000 visited configurations are compared to the current configuration.

Figure 3.7 shows the number of cycles detected when applying Tabucol on graph *dsjc500.5* with different values of the tabu tenure (referred to as Tl in the figure). For the entries $\alpha = 0.6$ and $\alpha = 1.0$, the tabu tenure is composed as specified in section 2.3, only modifying the value of α. All the other entries in the figure have a fixed tabu tenure, independent of any parameters. Figure 3.8 shows the best number of conflicts found by Tabucol using these values of the tabu tenure.

Comparing these two figures, it can be seen that while $\alpha = 0.6$ yields a quite large number of cycles, the resulting number of conflicts is quite good. On the other hand, when T_l is fixed to 5 or 10, the resulting number of conflicts is very high while the number of cycles that is detected is not that much greater than for $\alpha = 0.6$. When $\alpha = 1.0$, the number of cycles is much lower than for $\alpha = 0.6$ and the performance is also somewhat better.

Besides the number of cycles detected, we also know the distance between the cycles, that is the number of iterations between the current configuration and the configuration the current one is a cycle of. Figure 3.9 shows a histogram of these distances on graph *dsjc500.5*. What is interesting is that for $T_l = 5$ and $T_l = 10$ the number of cycles seems to be very constant over distance, while for the other parameters the chance of a cycle decreases as the distance increases.

Figure 3.7: The resulting number of cycles after applying Tabucol with different values of the tabu tenure on graph *dsjc500.5* with $k = 49$.

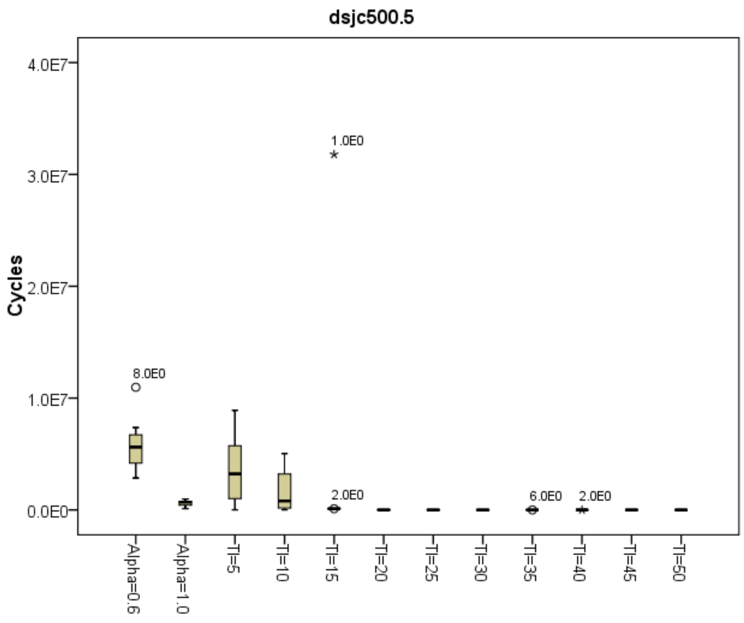


Figure 3.8: The resulting number of conflicts after applying Tabucol with different values of the tabu tenure on graph *dsjc500.5* with $k = 49$.

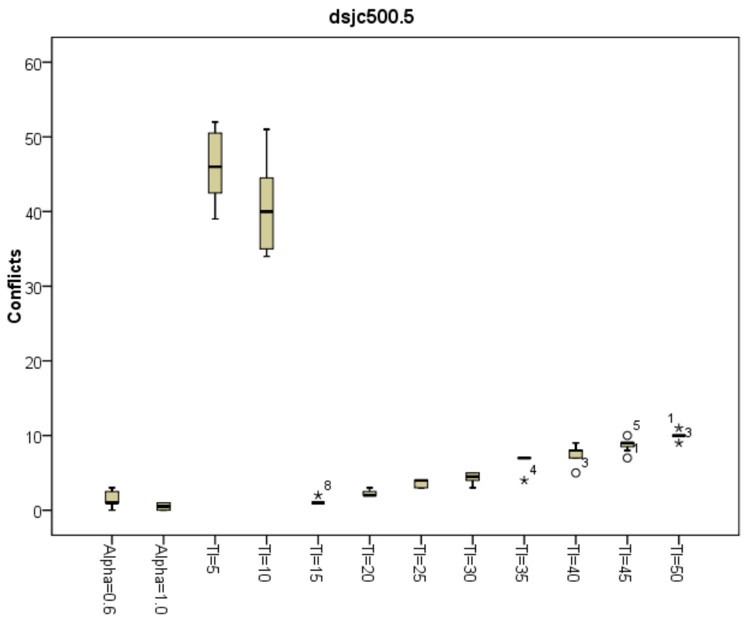
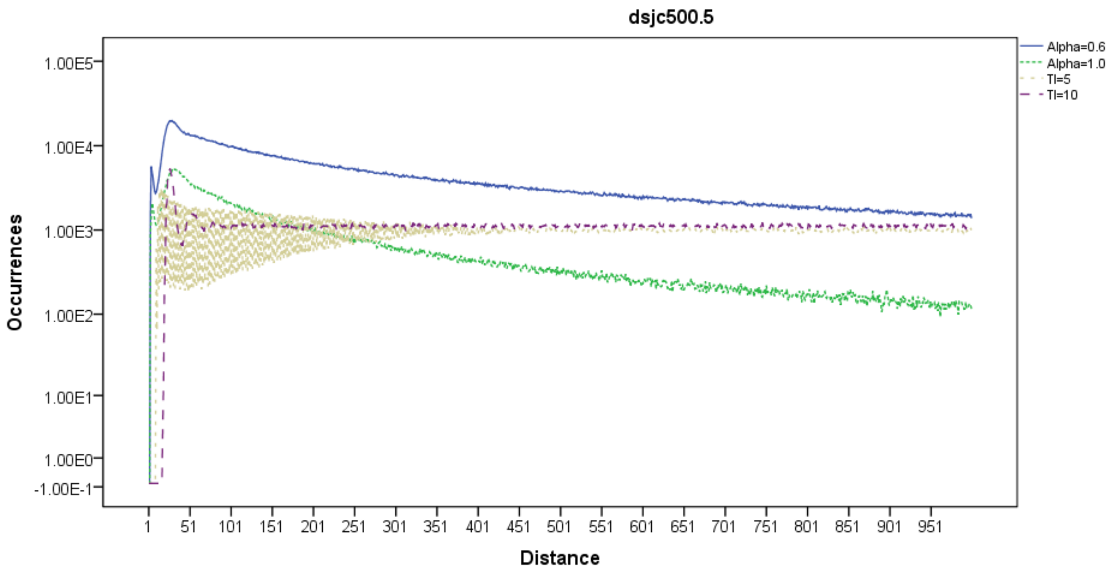
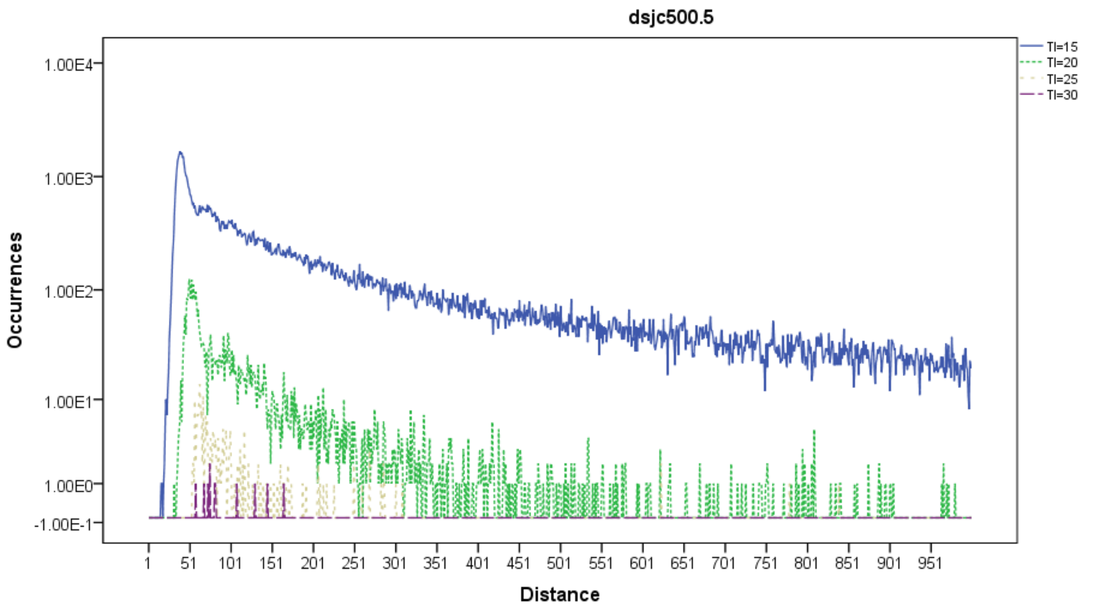


Figure 3.9: Histogram of the distances between cycles for different values of the tabu tenure on graph *dsjc500.5* with $k = 49$.





3.5.2 An alternative for detecting 'cycles'

The low number of cycles for the poor performing $T_l = 5$ and $T_l = 10$ in the previous section might be due to the limited number of configurations used for comparison. Another cause could be that the search algorithm does visit the same configurations again, but ones that are symmetric to the stored configuration and as such will not detected. Both these shortcomings can be lessened, it is possible to recognize symmetric configurations as the same just like PGTS does and the number of configurations used in the look-back can be increased. However, both these solutions increase the runtime even further.

As an alternative, we propose to use the following method: instead of storing previous configurations and comparing them to the current one, we only keep track of the number of moves we perform for each vertex, regardless of the color the move specifies. As the required extra runtime is negligible this can be kept for the entire search process.

Figure 3.10 shows the number of moves applied to each vertex for $T_l = 5$. As can be seen there is a very small number of vertices to which almost all the moves are applied (note the log scale). The other vertices are left mostly untouched. On the other hand, when using parameter $\alpha = 0.6$ the distribution of moves over the vertices looks like figure 3.11. When comparing these two figures, it is interesting to see that in 3.11 moves are relatively evenly distributed over the vertices and the resulting number of conflicts is good, while in 3.10 the moves are very unevenly distributed over the vertices and the resulting number of conflicts is very high. Concentrating moves on a few vertices appears to be very bad for performance. It is most likely that this is a indication of the algorithm being trapped in some region of the search space, without the ability to get out as moves are made tabu for such a short time only, and as such can be chosen over and over again.

Figure 3.10: Histogram of the number of moves for vertices graph *dsjc500.5* with $T_l = 5$ and $k = 49$.

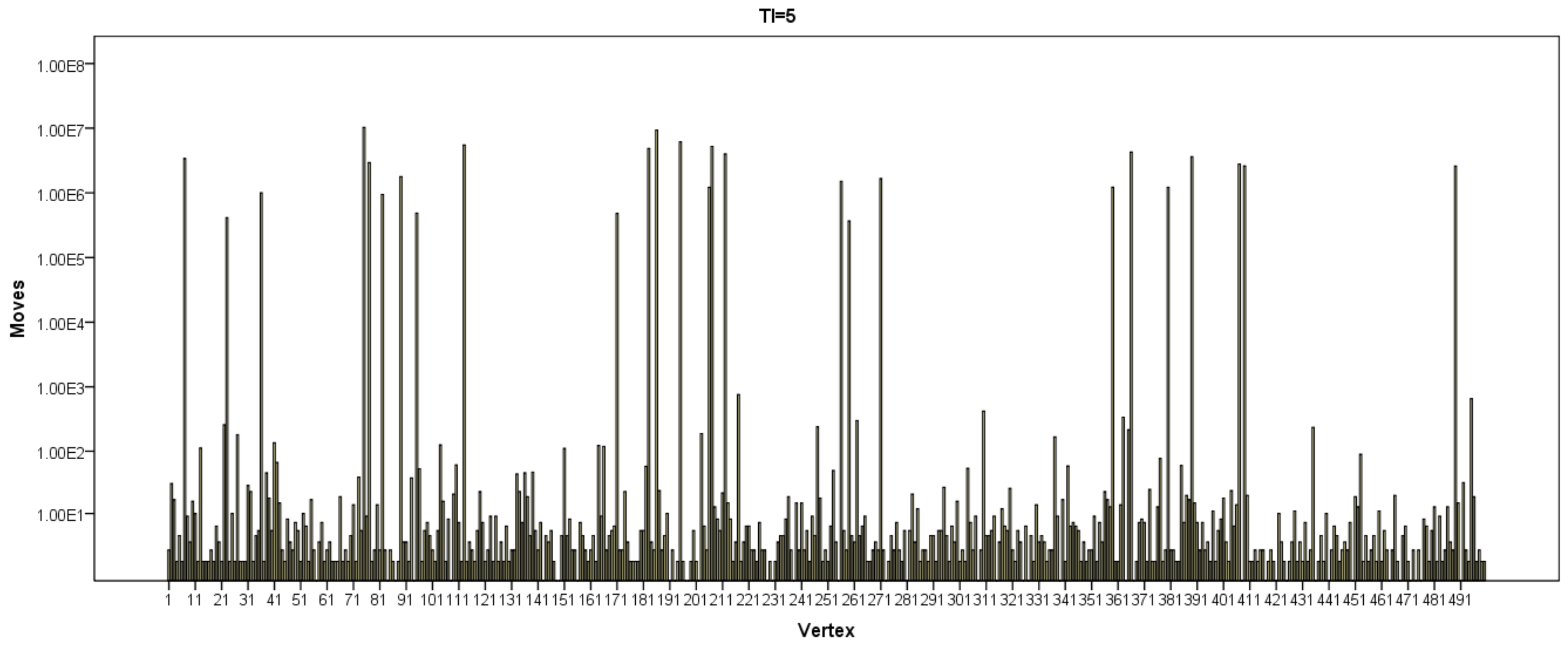
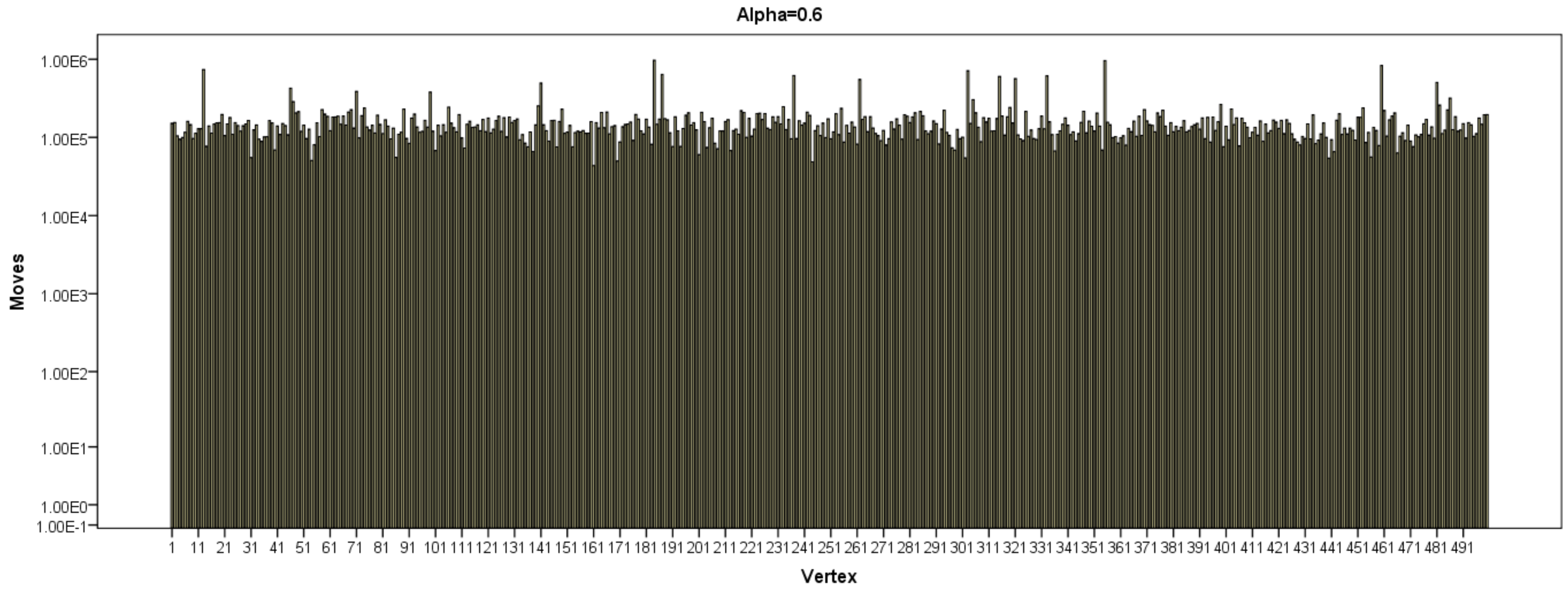


Figure 3.11: Histogram of the number of moves for vertices graph *dsjc500.5* with $\alpha = 0.6$ and $k = 49$.



3.6 Tabucol with fewer parameters

3.6.1 Simplifying the tabu tenure

In order to simplify the tabu tenure (and thus reduce the number of parameters of Tabucol) we have to decide which parameters are important to keep and which ones can be eliminated. As can be seen in section 3.4.1, the choice of Porumbel, Hao and Kuntz to make $\alpha = 0.6$ is questionable, as larger values for α like 0.9 or 1.2 often perform equally or better. This is why we propose to remove parameter α from the tabu tenure, which amounts to setting $\alpha = 1.0$. In section 3.4.3 it became clear that Tabucol is not very sensitive to variation of parameter *mmax*, and as such it is eliminated. In figure 3.9 it can be seen that a lot of cycles occur within a small distance (the initial spikes in the graph). These spikes are most likely due to the attraction of local optima, in which the search process gets lured back in. As these spikes seem especially prominent for non-static tabu tenures, it might be beneficial to specify a minimal value for the tabu tenure. Parameter A most likely fulfills this role, as in section 3.4.2 it can be seen that a small value for A results in much lower performance. Putting this together we propose the following simplified tabu tenure:

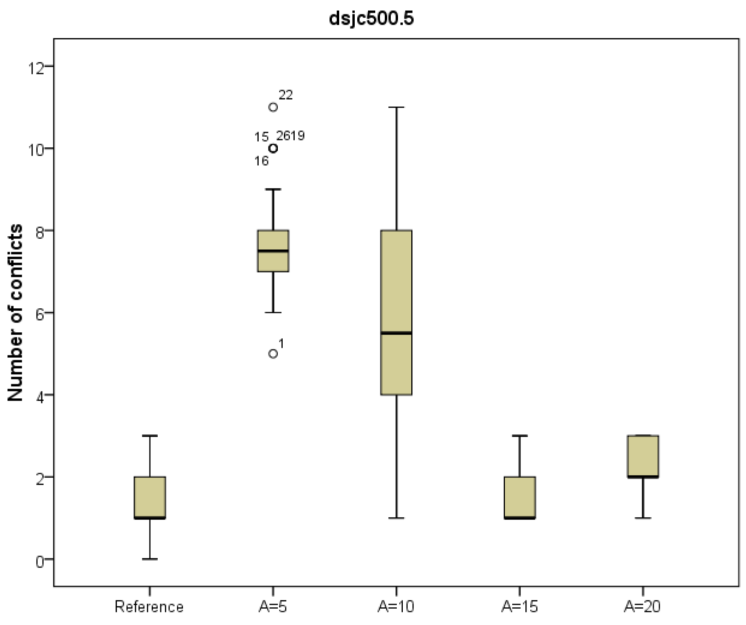
$$simpT_l = Max(f_c(C), A) \tag{3.2}$$

So the simplified tabu tenure is equal to the number of conflicts in the current configuration or the specified minimal value A, whichever is larger. So parameter A is still present, but compared to the tabu tenure of Porumbel, Hao and Kuntz (see equation 3.1) the minimal tabu tenure is absolute and not a random value from $\{0 \dots A\}$. This should reduce the chance of the algorithm getting trapped in local optima as moves are guaranteed not to be chosen for the number of iterations equal to A.

3.6.2 Performance of the simplified tabu tenure

In figure 3.12 the reference tabu tenure of Porumbel, Hao and Kuntz (as specified in section 2.3) with $\alpha = 0.6$, $A = 10$ and $mmax = 1000$, is compared to the simplified tabu tenure with different minimal values for graph *dsjc500.5*. Compared to the reference tabu tenure the simplified tabu tenure performs quite good for $A = 15$, although it never finds a legal solution. The results for the graphs *flat300_28_0* and *le450_25c* are comparable that those of *dsjc500.5* and can be found in appendix B. For graph *dsjc1000.1* the results look a bit different, as here the simplified tabu tenure performs better than the reference tabu tenure for $A = 20$ as can be seen in figure 3.13. In the case of graph *r1000.1c* this difference is even larger as can be seen in figure 3.14: whereas the reference tabu tenure leads to a legal solution a number of times, the simplified tabu tenure with $A = 20$ will find a legal solution in all test runs. In order to find out whether cycles are the cause of the performance difference between the reference tabu tenure and the simplified tabu tenure, two histograms were made of the number of moves applied to each vertex. Figure 3.15 shows the histogram for the reference tabu tenure and figure 3.16 the histogram for the simplified tabu tenure with $A = 20$. The difference does not seem to be extremely large, but for the simplified tabu tenure the moves appear to be more evenly distributed over the vertices. So the reference tabu tenure tends to focus more on certain moves

Figure 3.12: Number of conflicts for graph *dsjc500.5* using different minimal tabu tenures with $k = 49$.



than the simplified tabu tenure, which seems to indicate that the algorithm is stuck in a region in the search space.

3.6.3 Use of the simplified tabu tenure for PGTS

While the simplified tabu tenure appears to perform well for Tabucol, this is not necessarily the case for the extension PGTS. To find out whether the simplified tabu tenure might be useful for PGTS, tests were performed on the graphs *dsjc500.5*, *le450_25c* and *r1000.1c*. The runtime for each search instance was limited to one hour. The results can be seen in figures 3.17, 3.18 and 3.19. For graph *dsjc500.5* with $A = 15$, the results are comparable to the reference tabu tenure. For this graph the simplified tabu tenure works better with PGTS than with Tabucol, as the combination with Tabucol never found an optimal solution. For graph *le450_25c* the results of the simplified tabu tenure are comparable to the reference tabu tenure as well, using $A = 20$. For graph *r1000.1c*, results of the simplified tabu tenure with $A = 20$ are better than for the reference tabu tenure, as it is the case for Tabucol. Although only three graphs were tested, results seem to indicate that the simplified tabu tenure works just as well for PGTS as it does for Tabucol.

Figure 3.13: Number of conflicts for graph *dsjc1000.1* using different minimal tabu tenures with $k = 20$.

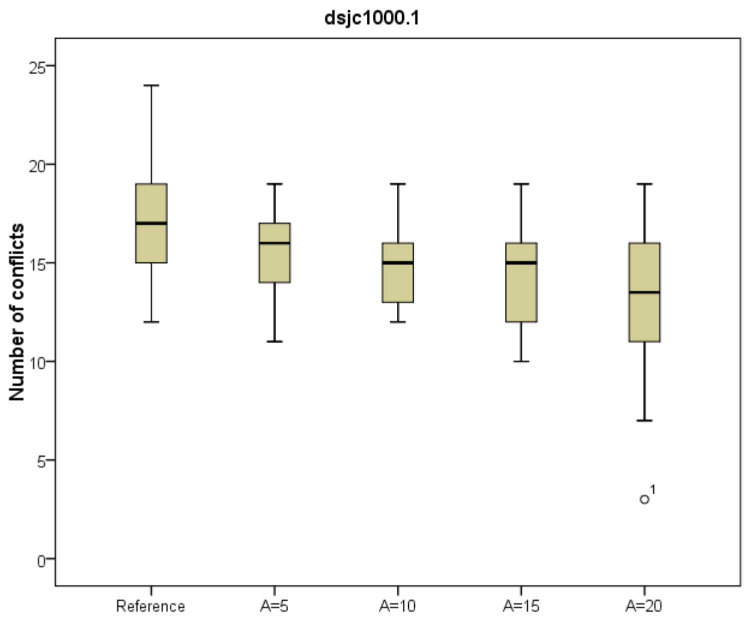


Figure 3.14: Number of conflicts for graph *r1000.1c* using different minimal tabu tenures with $k = 98$.

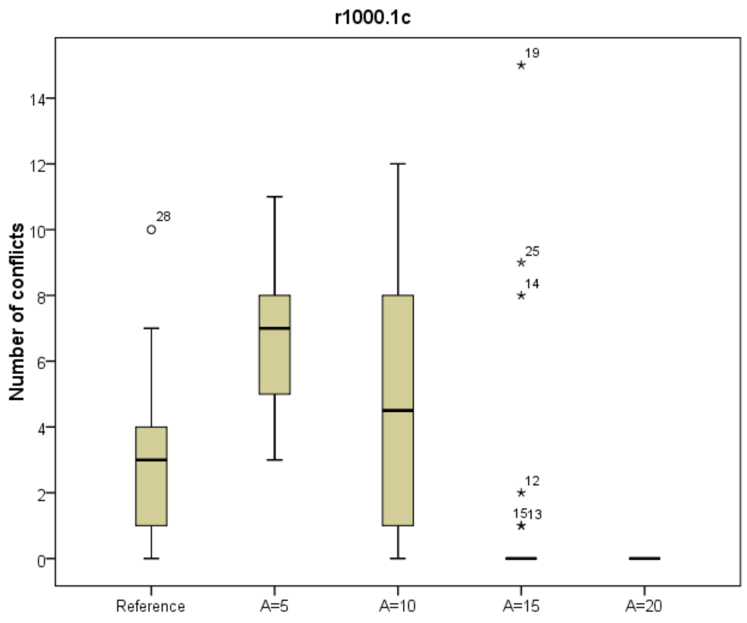


Figure 3.15: Histogram of the number of moves performed on each vertex for graph *r1000.1c* using the reference tabu tenure with $k = 98$.

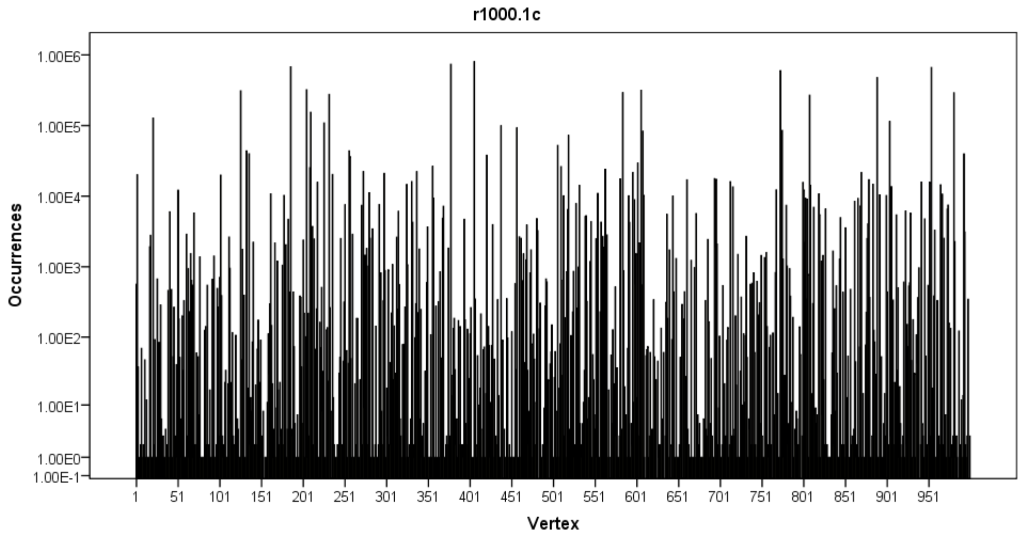


Figure 3.16: Histogram of the number of moves performed on each vertex for graph *r1000.1c* using the simplified tabu tenure with $A = 20$ and $k = 98$.

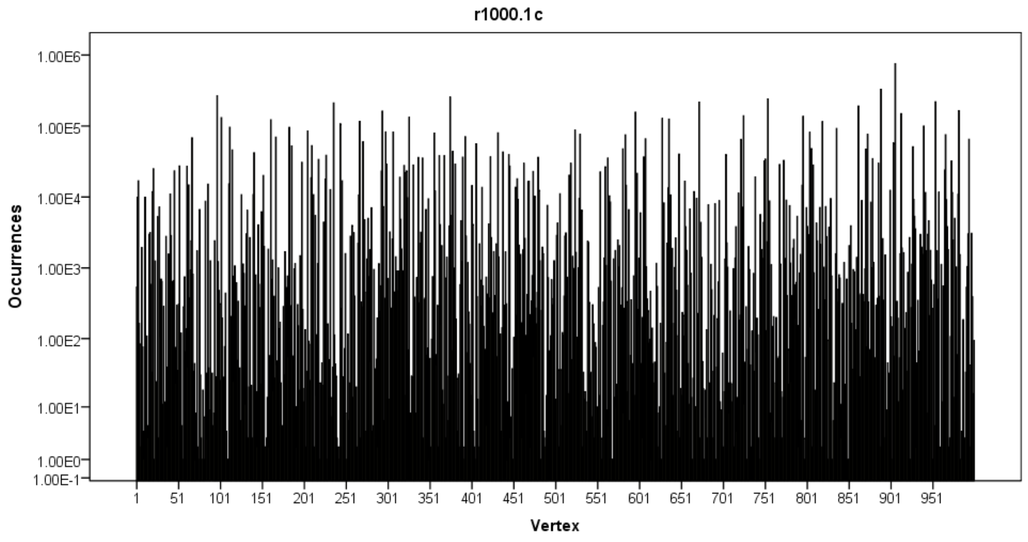


Figure 3.17: Number of conflicts for graph *dsjc500.5* using different minimal tabu tenures with PGTS and $k = 49$.

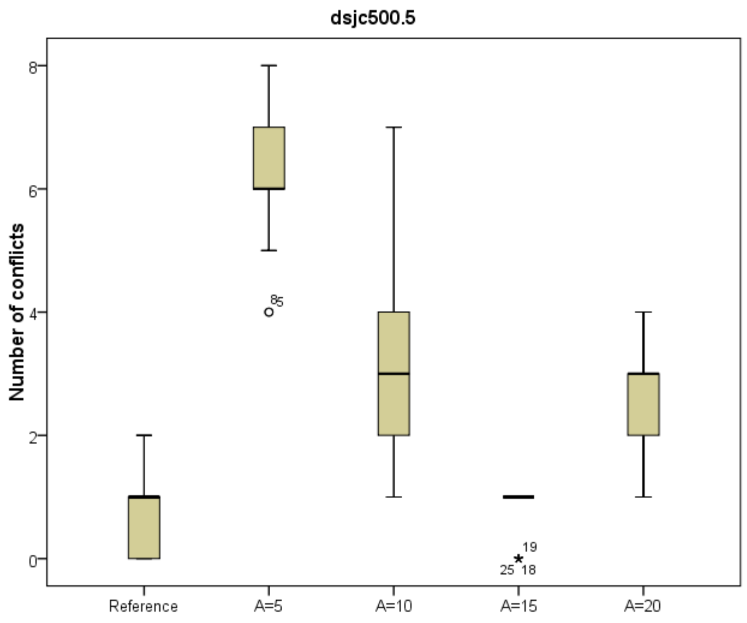


Figure 3.18: Number of conflicts for graph *le450_25c* using different minimal tabu tenures with PGTS and $k = 25$.

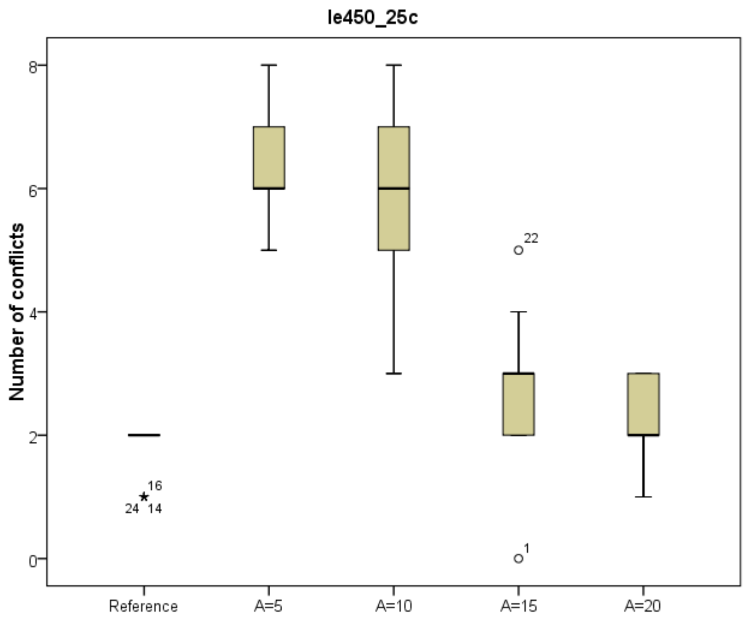
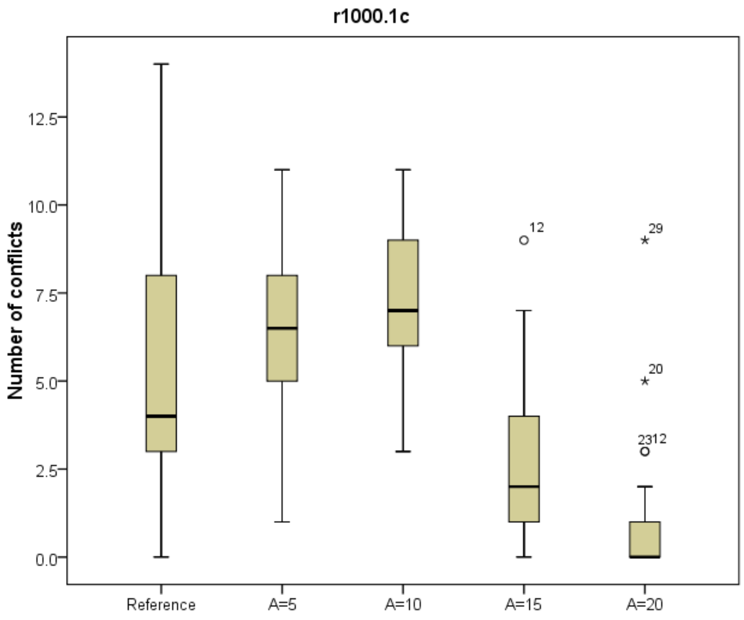


Figure 3.19: Number of conflicts for graph *r1000.1c* using different minimal tabu tenures with PGTS and $k = 98$.



Chapter 4

Discussion

While the simplified tabu tenure introduced in section 3.6.1 works well, it still requires the user to select a value for parameter A, which has a large influence on the performance of Tabucol. However, it might be possible to use the distribution of moves over the vertices to optimize this parameter automatically. As keeping track of the number of moves for each vertex has a negligible impact on the computational complexity of tabu search, there is no reason not to keep track of the number of moves per node. If the tabu tenure is chosen too small the distribution of moves over the vertices will be very uneven. It might be possible to exploit this and increase the minimal tabu tenure when the detected distribution of moves over the vertices is deemed too uneven. Unfortunately, this poses a new problem, as we now have to decide how to determine differences in the distribution of moves over the vertices and how large these differences should be before we decide they are too large. One possibility for measuring how moves are distributed over the vertices is to check for each vertex if the number of moves performed is larger than the median of the number of moves performed for all vertices. If there is a vertex for which it holds that the difference between the median of the number of moves for all vertices and the number of moves of this vertex is larger than some threshold we consider the distribution of moves over the vertices to be uneven. These check could be repeated at regular intervals, adjusting the tabu tenure when needed. Finding a good threshold value could be the subject of possible further research.

Outside the scope of tabu search for graph coloring, determining the distribution of moves over the search space can be a useful tool for every user wondering whether poor performance of a tabu search algorithm on some problem is the result of tabu search not exploring the entire search space, which can be due to a too small tabu tenure. As the computational overhead is really small, it is feasible to keep track of moves for each vertex for every run of a tabu search algorithm. If the results after a run are below expectation, it is possible to check whether the algorithm was focussing only on a part of the search space. If this is the case, the tabu tenure should be increased to avoid tabu search getting trapped in a local optimum. Users could optimize the search process by increasing the tabu tenure until the moves performed by tabu search are distributed evenly enough over the search space. Unfortunately this does not appear to work the other way round, i.e. being able to conclude that the tabu tenure is too large from the distribution of moves over the search space. But this can be avoided by starting out with a relatively small tabu tenure, checking the distribution of moves over the search space and adjusting the tabu tenure when needed. This is then repeated until the user is satisfied with the distribution of moves over the search space.

Chapter 5

Conclusion

We have investigated the importance and robustness of the parameters of Tabucol used for the tabu tenure and found that α and A have a very large influence on the performance, while *mmax* does not.

While cycles between visited configurations might decrease performance we have not been able to find a direct correlation between a large number of cycles and low performance. However, this might be due to limitations to detect them.

Furthermore we have found that small values of the tabu tenure, which are associated with low performance, lead the search algorithm to focus on moves of a select number of vertices, while larger values of the tabu tenure lead to a much more evenly distribution of moves over the vertices and a much better performance.

Lastly we have found that a simplified tabu tenure, which uses only the number of conflicts of the current solution (which eliminates the need of the Tabucol parameters α and *mmax*) and a minimal value (which is parameter A, but without the random component), performs close to the reference tabu tenure and even manages to improve upon it for the graphs *dsjc1000.1* and *r1000.1c*.

Further research could be done to investigate whether making the minimal tabu tenure adaptive leads to improved performance. If this parameter could be set by the algorithm itself it would remove yet another parameter having to be chosen be the user.

The simplified tabu tenure was only tested with PGTS for three graphs, but experiments on the graphs *dsjc500.5*, *le450_25c* and *r1000.1c* show favorable results compared to the reference tabu tenure.

We also proposed a tabu search technique which could be used as a tool by any tabu search user to determine if the tabu tenure is sufficiently large. This may help users to optimize the tabu search parameters such that the algorithm will not get stuck in regions of the search space by performing moves on part of the search space only.

Bibliography

[1] Cliques, coloring, and satisfiability: Second dimacs implementation challenge. *American Mathematical Society*, 1996.

[2] C. Avanthay, A. Hertz, and N. Zufferey. A variable neighborhood search for graph coloring. *European Journal of Operational Research*, 151(2):379–388, 2003.

[3] I. Blöchliger and N. Zufferey. A graph coloring heuristic using partial solutions and a reactive tabu scheme. *Computers and Operations Research*, 35(3):960–975, 2008.

[4] Daniel Brélaz. New methods to color the vertices of a graph. *Commun. ACM*, 22(4):251–256, 1979.

[5] M. Chams, A. Hertz, and D. de Werra. Some experiments with simulated annealing for coloring graphs. *European Journal of Operational Research*, 32(2):260–266, November 1987.

[6] Daniel Costa, Alain Hertz, and Clivier Dubuis. Embedding a sequential procedure within an evolutionary algorithm for coloring problems in graphs. *Journal of Heuristics*, Volume 1:105 – 128, 1995.

[7] L. Davis. Order-based genetic algorithms and the graph coloring problem. *Handbook of Genetic Algorithms*, pages 72–90, 1991.

[8] R. Dorne and J.K. Hao. A new genetic local search algorithm for graph coloring. *Lecture notes in computer science*, 1498/1998:745–754, 1998.

[9] C. Fleurent and J.A. Ferland. Genetic and hybrid algorithms for graph coloring. *Annals of Operations Research*, 63(3):437–461, 1996.

[10] C. Friden, A. Hertz, and D. de Werra. Stabulus: a technique for finding stable sets in large graphs with tabu search. *Computing*, 42(1):35–44, 1989.

[11] P. Galinier, A. Hertz, and N. Zufferey. An adaptive memory algorithm for the k-coloring problem. *Discrete Applied Mathematics*, 156(2):267–279, 2008.

[12] Philippe Galinier and Alain Hertz. A survey of local search methods for graph coloring. *Comput. Oper. Res.*, 33(9):2547–2562, 2006.

[13] F. Glover, C. McMillan, and B. Novick. Interactive decision software and computer graphics for architectural and space planning. *Annals of Operations Research*, 5(3):557–573, 1985.

[14] P. Hansen and N. Mladenović. Variable neighborhood search: Principles and applications. *European Journal of Operational Research*, 130(3):449–467, 2001.

[15] JK Hao and P. Galinier. Hybrid evolutionary algorithms for graph coloring. *Journal of Combinatorial Optimization*, 3(4):379–397, 1999.

[16] A. Hertz and D. Werra. Using tabu search techniques for graph coloring. *Computing*, 39(4):345–351, 1987.

[17] David S. Johnson, Cecilia R. Aragon, Lyle A. McGeoch, and Catherine Schevon. Optimization by simulated annealing: An experimental evaluation; part ii, graph coloring and number partitioning. *Operations Research*, 39(3):378–406, 1991.

[18] R. Karp. Reducibility among combinatorial problems. In R. Miller and J. Thatcher, editors, *Complexity of Computer Computations*, pages 85–103. Plenum Press, 1972.

[19] H.W. Kuhn. The hungarian method for the assignment problem. *Naval Research Logistic Quarterly*, 2:83–97, 1955.

[20] F.T. Leighton. A graph coloring algorithm for large cheduling problems. *Journal of Research of the National bureau of Standards*, 84:489–503, 1979.

[21] Zhipeng Lü and Jin-Kao Hao. A memetic algorithm for graph coloring. *European Journal of Operational Research*, 203(1):241–250, May 2010.

[22] C. Morgenstern and H. Shapiro. Chromatic number approximation using simulated annealing. *Unpublished manuscript*, 1986.

[23] Craig Morgenstern. Distributed coloration neighborhood search. cliques, coloring and satisfiability. *Second DIMACS Implementation Challenge*, 1993.

[24] Daniel Cosmin Porumbel, Jin-Kao Hao, and Pascale Kuntz. Position-guided tabu search algorithm for the graph coloring problem. In Thomas Stützle, editor, *LION*, volume 5851 of *Lecture Notes in Computer Science*, pages 148–162. Springer, 2009.

[25] Daniel Cosmin Porumbel, Jin Kao Hao, and Pascale Kuntz. A search space “cartography” for guiding graph coloring heuristics. In *International Conferences*, pages 769–778, 2009.

[26] Daniel Cosmin Porumbel, Jin-Kao Hao, and Pascale Kuntz. An evolutionary approach with diversity guarantee and well-informed grouping recombination for graph. *Computers and Operations Research*, 37(10):1822–1832, 2010.

[27] Y. Rochat and É.D. Taillard. Probabilistic diversification and intensification in local search for vehicle routing. *Journal of heuristics*, 1(1):147–167, 1995.

Appendix A

Parameter robustness

A.1 Parameter α

In figures A.1, A.2 and A.3 the parameter *alpha* is varied for runs of Tabucol. It can be seen that the best results are attained for the larger values of α, 0.9 and 1.2.

A.2 Parameter A

In figures A.4, A.5 and A.6 the parameter A is varied for runs of Tabucol. It can be seen that the best results are attained for the larger values of A, 20 and 30.

A.3 Parameter $mmax$

In figures A.7, A.8 and A.9 the parameter $mmax$ is varied for runs of Tabucol. Differences in performance between different values of the parameter appear to be minimal.

Figure A.1: The resulting number of conflicts after applying Tabucol with different values of parameter α on graph *dsjc1000.1* with $k = 20$.

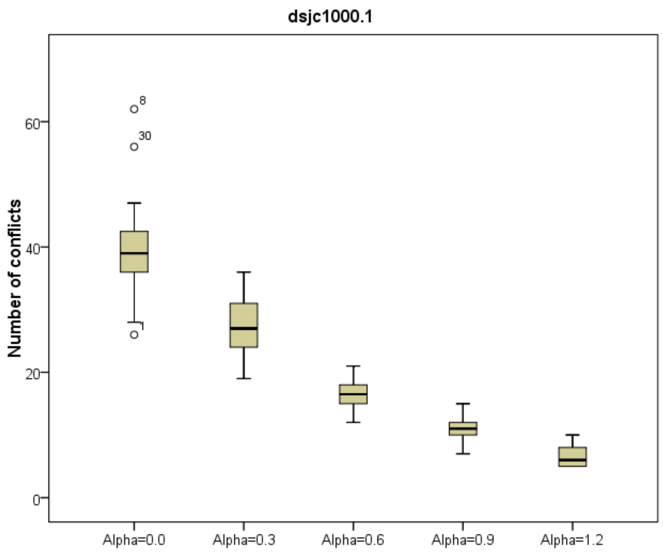


Figure A.2: The resulting number of conflicts after applying Tabucol with different values of parameter α on graph *le450_25c* with $k = 25$.

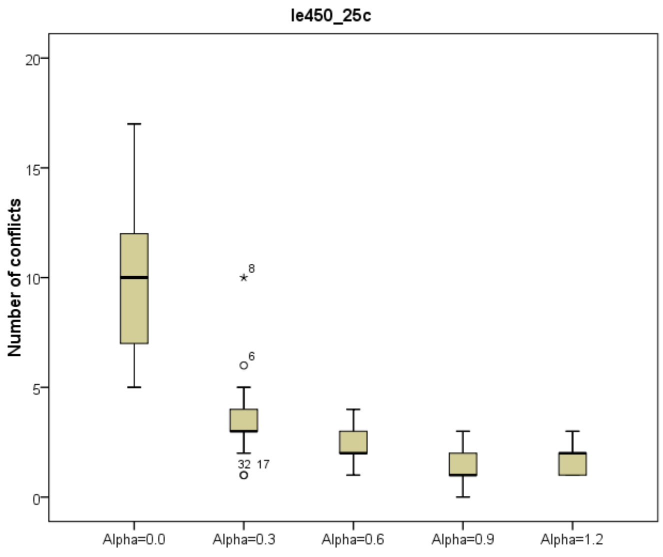


Figure A.3: The resulting number of conflicts after applying Tabucol with different values of parameter α on graph *r10001.c* with $k = 98$.

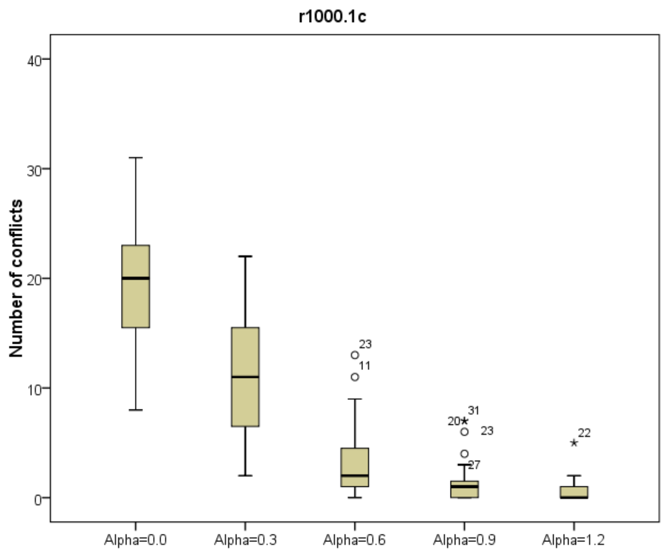


Figure A.4: The resulting number of conflicts after applying Tabucol with different values of parameter A on graph *dsjc1000.1* with $k = 20$.

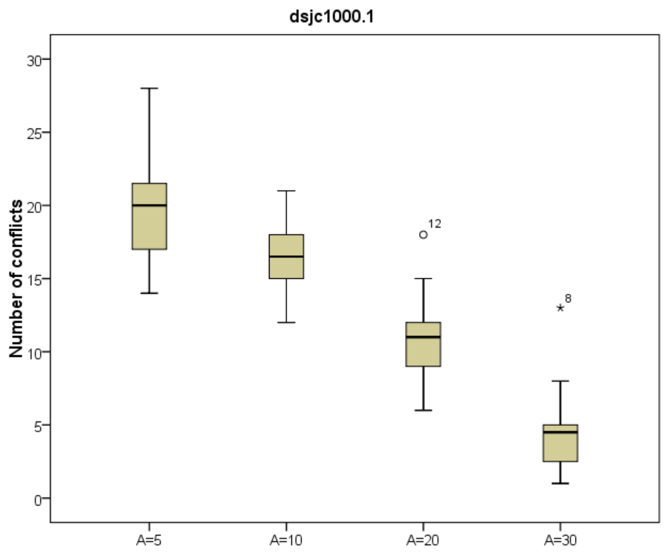


Figure A.5: The resulting number of conflicts after applying Tabucol with different values of parameter A on graph *le450_25c* with $k = 25$.

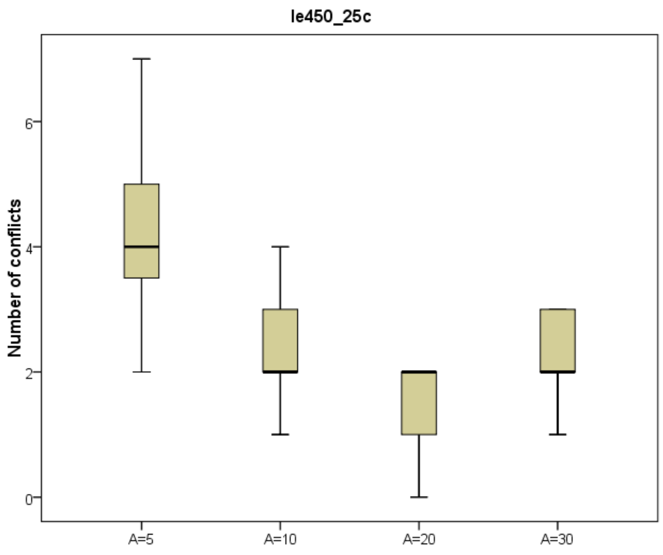


Figure A.6: The resulting number of conflicts after applying Tabucol with different values of parameter A on graph *r1000.1c* with $k = 98$.

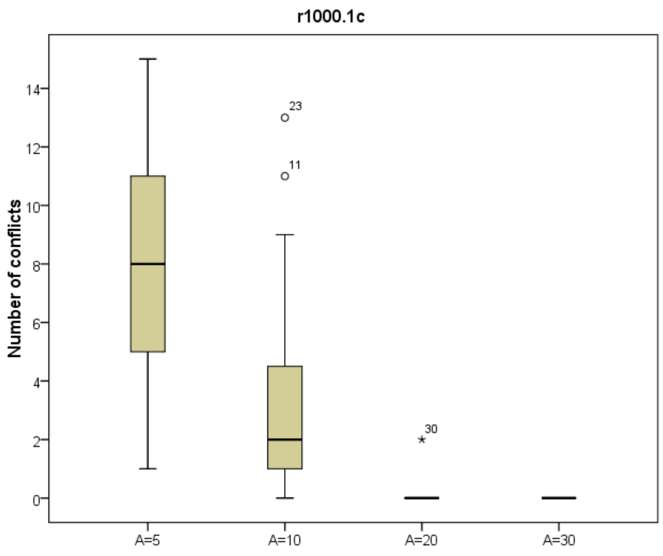


Figure A.7: The resulting number of conflicts after applying Tabucol with different values of parameter *mmax* on graph *dsjc1000.1* with $k = 20$.

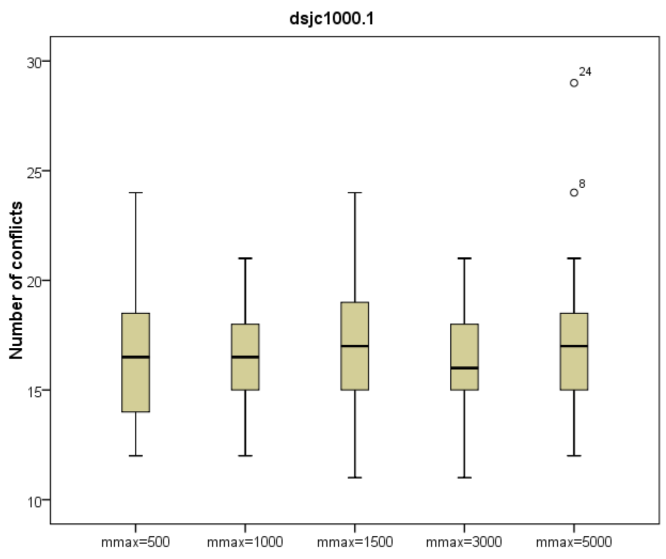


Figure A.8: The resulting number of conflicts after applying Tabucol with different values of parameter *mmax* on graph *le450_25c* with $k = 25$.

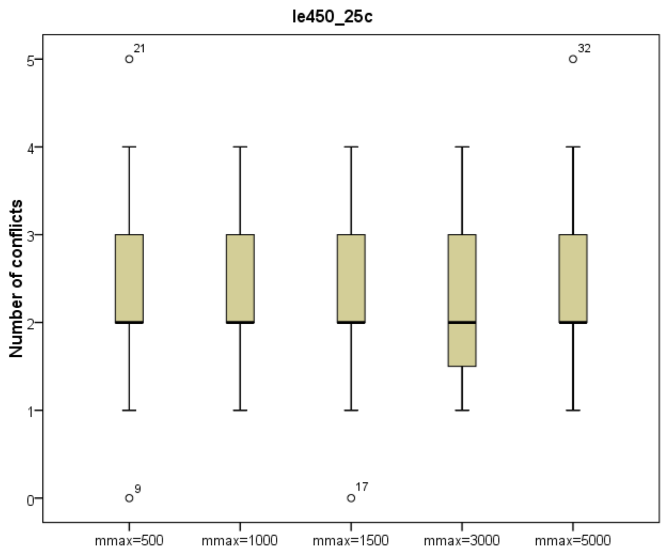
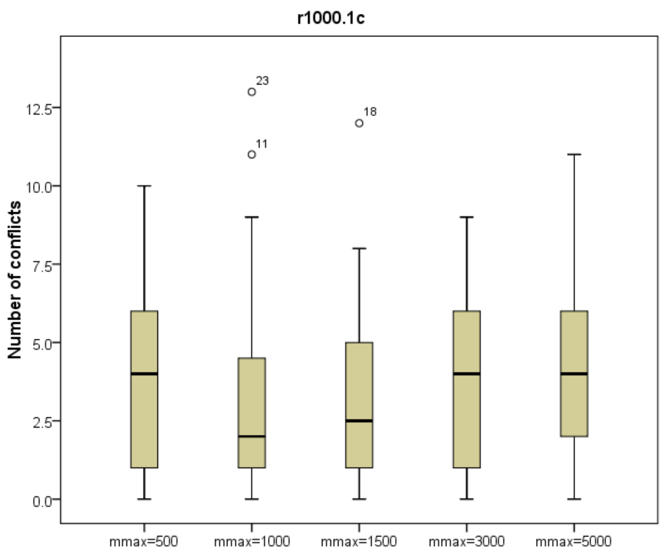


Figure A.9: The resulting number of conflicts after applying Tabucol with different values of parameter *mmax* on graph *r1000.1c* with $k = 98$.



Appendix B

Simplified tabu tenure

The figures B.1 and B.2 show a comparison of performance of Tabucol using the default tabu tenure of section 2.3 and the simplified tabu tenure as defined in section 3.6.1 with different minimal tabu tenures. For the graph *flat300_28_0*, performance is very similar for all the tested parameters, while for graph *le450_25c*, the simplified tabu tenure with a minimal tabu tenure of 20 performs comparable to the reference tabu tenure.

Figure B.1: Number of conflicts for graph *flat300_28_0* using different minimal tabu tenures and $k = 29$.

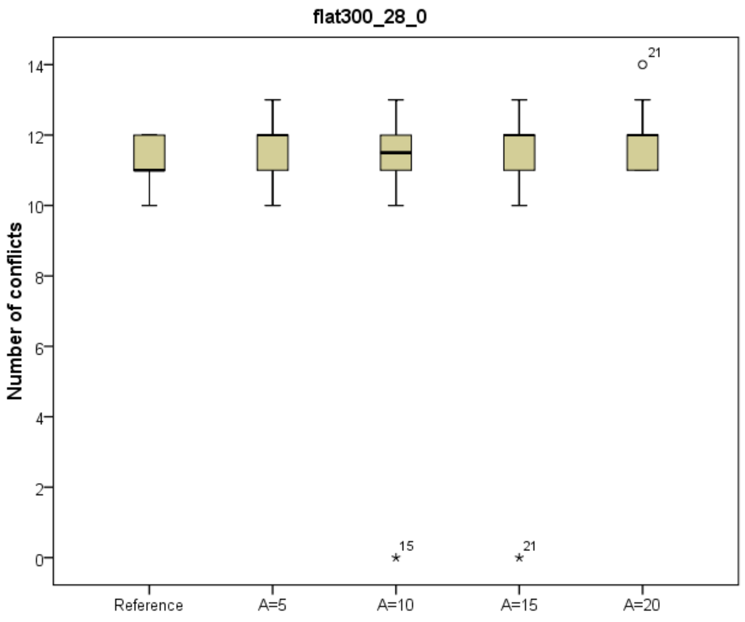
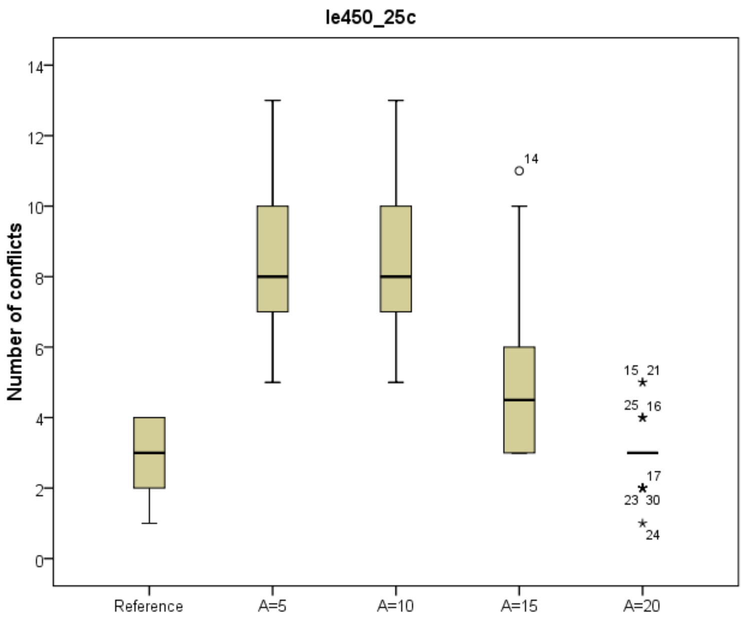


Figure B.2: Number of conflicts for graph *le450_25c* using different minimal tabu tenures and $k = 25$.


le450_25c
Number of conflicts
Reference
A=5
A=10
A=15
A=20