Population-based and Learning-based Metaheuristic Algorithms for the Graph Coloring Problem

David Chalupa
Institute of Applied Informatics
Faculty of Informatics and Information Technologies
Slovak University of Technology
Ilkovicova 3
84216 Bratislava, Slovakia
chalupa.david@gmail.com

ABSTRACT

In this paper, two new metaheuristic algorithms for the graph coloring problem are introduced. The first one is a population-based multiagent evolutionary algorithm (MEA), using a multiagent system, where an agent represents a tabu search procedure. Rather than using a single long-term local search procedure, it uses more agents representing short-term local search procedures. Instead of a specific crossover, MEA uses relatively general mechanisms from artificial life, such as lifespans and elite list [3, 4]. We are introducing and investigating a new parametrization system, along with a mechanism of reward and punishment for agents according to change in their fitness. The second algorithm is a pseudo-reactive tabu search (PRTS), introducing a new online learning strategy to balance its own parameter settings. Basically, it is inspired by the idea to learn tabu tenure parameters instead of using constants. Both algorithms empirically outperform basic tabu search algorithm TabuCol [8] on the well-established DIMACS instances [10]. However, they achieve this by using different strategies. This indeed shows a difference in potential of population-based and learning-based graph coloring metaheuristics.

Categories and Subject Descriptors

I.2.6 [**Learning**]: Parameter learning; I.2.8 [**Problem Solving, Control Methods, and Search**]: Heuristic methods; I.2.11 [**Distributed Artificial Intelligence**]: Multiagent systems

General Terms

Algorithms, Experimentation

Keywords

Graph Coloring, Tabu Search, Metaheuristics, Multiagent Systems, Pseudo-reactive Tabu Search, Parameter Learning

1. INTRODUCTION

Let $G = [V, E]$ be an undirected graph and let c be a number of colors. The objective of the graph coloring problem is to find a partitioning of the vertex set V into color classes $V_1, V_2, ..., V_c$ such that:

1. the color classes cover the whole vertex set: $V_1 \cup V_2 \cup ... \cup V_c = V$ and
2. the number of adjacent vertices in the same color classes is minimal: $J = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} confl(i,j) \rightarrow \min$, where

$$confl(i,j) = \begin{cases} 1 & \exists k : v_i \in V_k \wedge v_j \in V_k \wedge [v_i, v_j] \in E \\ 0 & otherwise \end{cases} \quad (1)$$

In the terms of evolutionary computation, the traditional approach is based on maximization of a so-called *fitness function*. Due to this fact, we state the problem as maximization of the following fitness function: $F = |E| - J$. If two connected vertices are equally colored, we will refer to this situation as a *collision*. The minimal number of colors needed to color a graph with no collisions is called *chromatic number* and denoted as χ.

2. RELATED WORK

The most traditional algorithms are the *constructive heuristics* that simply build the solution iteratively. Usually, greedy approach is used to determine vertex and color for particular iteration. The most popular constructive algorithm is the *Brélaz heuristic*, using degrees of vertices and their current saturations[1] [2].

Another well-established approach is the *local search*. Local search algorithms iteratively try to improve fitness by using small modifications called *moves* or *mutations*. In a majority of current local search graph coloring methods, if a move is performed, an inverse move is forbidden for several iterations to prevent cycling. This approach is known

[1]Saturation of a vertex is the number of its neighboring vertices which are already colored.

as *tabu search*. The number of iterations, for which a move is forbidden, is called *tabu tenure*. The most popular tabu search algorithm for the graph coloring is TabuCol, introduced by Hertz and De Werra [8]. This algorithm iterates over colorings, where each vertex is colored and simply minimizes the number of collisions. TabuCol is widely used as a subroutine in various hybrid evolutionary algorithms [5, 7, 11, 14]. Another interesting tabu search algorithm is called PartialCol, which uses representation initially proposed by Morgenstern [13]. The idea is to use partial legal colorings of subgraphs instead of coloring every vertex. For more detailed information about this approach, reader may refer to paper by Blöchliger and Zufferey [1].

The most successful heuristics up-to-date are hybrid evolutionary algorithms, usually combining a crossover and a local search procedure [5, 7, 11, 14]. First very strong hybrid algorithms were proposed by Dorne and Hao [5] and Galinier and Hao [7] that used a greedy procedure to recombine color classes from two parents. Another interesting example of hybrid evolutionary approach is the EvoCol algorithm by Porumbel, et al. [14]. This method uses a partition-based crossover of multiple parents and a diversification strategy based on distances between individuals in the state space.

The first approach we propose, a multiagent evolutionary algorithm (MEA), is basically inspired by an experimental algorithm proposed by Comellas and Martinez-Navarro, called the Bumblebees algorithm [4]. This approach uses a simulation of bumblebees, collecting food in an artificial world, and introduces several mechanisms, which we further developed in our work, such as the lifespan, agent's birth and the elite list. In fact, MEA was developed from a synthesis of Bumblebees algorithm with TabuCol. The Bumblebees algorithm was also developed from another multiagent approach, using a mixed population of eternal and mortal agents, called Angels and Mortals [3].

3. BASIC TABU SEARCH FOR THE GRAPH COLORING PROBLEM

As we have already mentioned, TabuCol was the first and still is the most popular tabu search algorithm for the graph coloring problem. Because this algorithm represents a groundstone for both algorithms proposed in this paper, we will further describe it in this section. Basically, TabuCol starts with an initial coloring (usually a random assignment of colors to vertices) and improves it iteratively with mutations [8].

Suppose that we have an actual coloring S, on which we are going to perform a mutation. A *conflicting vertex* is a vertex, which is involved in at least one collision. A *neighborhood* $N(S)$ is then defined as a set of all colorings obtained from S by recoloring every conflicting vertex with every other possible color. From these colorings, a "training" subset V^* is chosen, each of the colorings in V^* is evaluated by the fitness function and the one with the highest fitness is chosen as a new actual coloring S^*. This process is iteratively repeated until a maximal number of iterations is reached or an optimal coloring is found. Note that if a tabu move leads to a coloring that has a higher fitness than anything else found so far, tabu search accepts it, despite the fact that it is tabu. This is called the *aspiration condition* [8].

When $V^* = N(S)$, the neighborhood construction can be done systematically, we will later refer to this as to the *precise approach*. When $V^* \neq N(S)$, a subset of colorings with predefined size must be randomly chosen. We will refer to this as to the *lazy approach*.

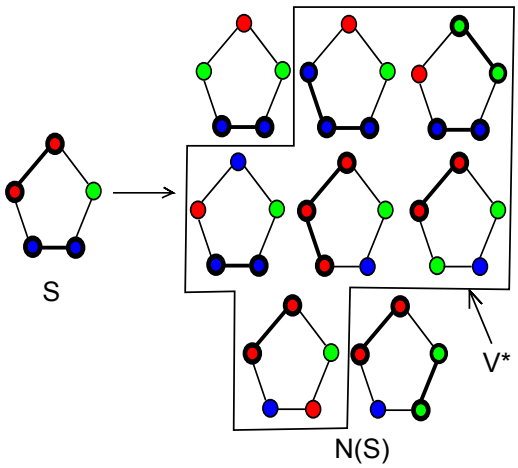

Figure 1: The neighborhood.

When we perform these moves, it is quite easy to get trapped, cycling in an uninteresting region of the state space. Suppose that vertex v had a color c_1 and the search algorithm recolored it with c_2. Thus, the move is defined by the couple $[v, c_2]$. However, the inverse move $[v, c_1]$ would lead the search back to the previous state. Thus, the tabu search forbids this move for a number of iterations defined by the tabu tenure [8].

Generally, it has been verified that static definition of the tabu tenure by a constant is ineffective. Dorne and Hao have proposed a dynamic tabu tenure, using the number of conflicting vertices, i.e. the number of vertices meaningful to recolor, as a metric for the calculation [5]:

$$t_l = \alpha f(S) + random(0, A-1), \tag{2}$$

where α and A are constants, $random(a, b)$ returns quasirandom integer between a and b and $f(S)$ denotes the number of conflicting vertices in S. The α and A constants are usually set to $\alpha = 0.6$ and $A = 10$ as proposed by Galinier and Hao [7]. This scheme was later called a DYN scheme [1]. It is by far the most widely used tabu scheme in current graph coloring literature.

4. MULTIAGENT EVOLUTIONARY ALGORITHM

The multiagent system we propose works with a population of individuals, each representing a potential solution to the problem. From this point of view, this basic concept is quite similar to traditional genetic algorithms. However, our method is only mutation-based, it does not need any crossover operator. The mutation mechanism is encapsulated in a procedure of local search. This allows us to preserve general nature of the multiagent system, there is no need for problem-specific knowledge in this general framework. All the specific information needed for the graph coloring problem is located only in the formulation of the local search.

4.1 General framework

The process starts with an initialization of the population. In operation *generate_agents()*, we generate $|P_0|$ agents, each representing an entirely random coloring. In the beginning, each agent is assigned a *lifespan*. The initial value of lifespan is given simply by parameter q of our model.

The initialization is followed by an iterative procedure, consisting of generations. In each generation, we first check for an optimal solution in the population. Then, in operation *decrease_lifespans()*, we decrement lifespan of each agent by a single unit. If an agent reaches zero lifespan, it is eliminated in procedure *eliminate()*. Step *agent's_birth()* is performed only each T_b generations, where T_b is a parameter of our algorithm. In this step, a new agent is born and a state from a so-called elite list is assigned to it. Agent's birth does not explicitly use any other information from the previous agents. Next, each agent performs *local_search()* for a certain number of iterations. In the last operation of *evaluation()*, lifespan of an agent is modified according to change in the fitness. The whole multiagent approach is described by the pseudocode of Algorithm 1.

At this point, let us further look on the operations of elimination and birth. Our algorithm uses a data structure called *elite list* to record several states previously found by eliminated agents. These states are then assigned to newly born agents. In fact, this means that the algorithm does restarts from some previous partially optimized states.

The elite list is not just a data structure to record the best states found in the search process. It is managed by the following rules:

1. Each agent can pass only one state to the elite list per its life and this happens at the time of its elimination.
2. State, which is passed to the elite list, must be a result of local search in some generation. It cannot be an intermediate result.
3. Passed state is accepted by the elite list only if its fitness is at least equal to fitness of the worst state in the elite list. If there are more of worst states, the one to be replaced is chosen quasirandomly.

First, we explain the intuitive view of the elite list management conditions. First condition means that an agent cannot pass more than one state to the elite list. This means that even when a single agent is able to "fly away" and suddenly find a path to states with higher fitness than any other agent, it is not allowed to pass all these states to the elite list, thus preserving natural diversity in the elite list. The second condition is illustrated by Figure 2. Suppose that this is a curve generated by fitness function over an agent's life. The vertical lines illustrate the moments, when the local search in a generation finishes. We will refer to them as so-called *milestones*. Restricting the set of states eligible for the elite list to the milestones simply reduces the risk of restarting the local search in a local optimum. In our preliminary experiments, this was shown to be an efficient strategy when dealing with local optima - not to overcome them but to evade them when doing a restart.

One may probably also ask, why is the search restarted in a previously visited state, when an agent is born. Graph coloring instances tend to generate very large plateaux, on which tabu search performs randomly. This is due to the fact that neighborhood may contain multiple states with highest fitness. These restarts, although very simple in concept, provide much help to search in promising regions more precisely.

Figure 2: The milestones.

In the next part, we will discuss the tabu search procedure, performed in the operation of local search. However, after the local search finishes, we compare fitness of the initial state and final state obtained after the search in the evaluation operation. If the final state has higher fitness than the initial, agent gains r units of lifespan. On the other hand, if the final state has lower or equal fitness as the initial, agent loses r units of lifespan. The value of r is a parameter of our algorithm.

4.2 Tabu search subroutine

In the procedure of local search, our algorithm simply performs TabuCol for each agent with the dynamic tabu tenure (with $\alpha = 0.6$ and $A = 10$), as we already mentioned in Section 3. The maximal number of iterations of tabu search that an agent performs in a single generation, is given by the following formula:

$$t_{max} = \beta \frac{c|V|}{|V^*|}. \tag{3}$$

The number of colors c and the number of vertices $|V|$ are used as metrics of the instance size, to provide more iterations for larger instances. On the other hand, the higher the "training" neighborhood size $|V^*|$ is, the lower number of iterations is performed. This is useful not only to balance the computational demands of different agents but it also helps to extend the search when being very close to the global optimum. Parameter β is used to tune the performance. Note that for this calculation, we use the neighborhood size $|V^*|$ at the beginning of the tabu search in a generation.

We use both the precise and lazy approach, as we mentioned in Section 3. Percentage of precise agents is denoted by parameter p of our algorithm. When doing the lazy tabu search, the "training" neighborhood size $|V^*|$ is fixed in the beginning of a particular generation by the following formula:

$$|V^*| = random(f(S), (c-1)f(S)). \tag{4}$$

The amount of conflicting vertices $f(S)$ is used as the lower bound for $|V^*|$ to avoid using extremely small neighborhoods that cause rapid reduction of fitness. The value $(c-1)f(S)$ is the size of the largest meaningful neighborhood, equal to $|N(S)|$. In the case of precise approach, $V^* = N(S)$, so one does not have to define $|V^*|$ explicitly.

5. PSEUDO-REACTIVE TABU SEARCH

The second algorithm, presented in this paper, is a pseudo-reactive tabu search. It is basically inspired by the *reactive*

Algorithm 1: The multiagent optimization model

	The multiagent optimization model
1	generate_agents()
2	for $g = 1..g_{max}$
3	best_agent = find_best_agent()
4	if best_agent.fitness = optimal_fitness
5	return S = best_agent.state
6	decrease_lifespans()
7	eliminate()
8	agent's_birth()
9	local_search()
10	evaluation()
11	return S = best_state_ever_found()

tabu search that observes fluctuation of the fitness function for a certain period of time and increments the tabu tenure, when the search seems to be trapped according to this fluctuation. Typical example of a reactive tabu scheme is the FOO scheme introduced by Blöchliger and Zufferey [1].

Pseudo-reactive approach is based on a similar idea. However, instead of directly incrementing the tabu tenure, it changes parameters, from which the tenure is calculated. Tabu tenure is then influenced indirectly, providing advantages of both reactive tabu tenure and the dynamic tabu tenure used in MEA.

5.1 Pseudo-reactive tabu tenure

Our pseudo-reactive tabu tenure is calculated with a very similar formula, as the dynamic tabu tenure:

$$t_l = \frac{\alpha_T}{10} f(S) + A, \tag{5}$$

where A is no longer a parameter of randomization (see Section 3), but simply an increment of the tabu tenure.

Parameters α and A are no longer fixed, but restricted only by predefined intervals, in which the algorithm is able to move. Instead of $\alpha_T = 6$, we use $\alpha_T \in \{0, 1, 2, ..., 19, 20\}$ and instead of $A = 10$, we use $A \in \{0, 1, 2, ..., 49, 50\}$.

In the beginning, we generate α and A with uniform distribution over these intervals. Then, we observe behavior of fitness during periods of ϕ iterations, similarly as in the reactive approach [1]. We take the highest value F_h and lowest value F_l of fitness during this period and evaluate difference $F_h - F_l$. We define two thresholds: b and c. If $F_h - F_l \leq b$, the search seems to be trapped and we change the parameters to provide longer tabu tenure. On the other hand, when $F_h + c \leq F_{BEST}$, where F_{BEST} is fitness of the best coloring found so far, the search may be unable to find high-quality colorings due to long tabu tenure, so we change the parameters to provide shorter tabu tenure. With certain probability p_{mut}, parameters are also changed even when none of this two conditions is fulfilled.

5.2 Parameter mutations

If $F_h - F_l \leq b$, we randomly choose one of the 5 parameters to change (see Table 1). Value of this parameter is changed by a mutation. Parameters α_T, A, b are mutated by an increment and ϕ and c are mutated by a decrement in this case. The mutation value is uniformly generated according to Table 1 (mutation columns).

If $F_h + c \leq F_{BEST}$, the process is the same. However, the mutation "directions" are reversed. Parameters α_T, A, b are mutated by a decrement and ϕ and c are mutated by an increment. The intervals for mutation values are also given by Table 1.

Using this strategy, the algorithm is provided by a simple but efficient mechanism of self-balancing for the parameter values. While α and A influence the tabu tenure directly, b and c influence the observation, whether the search is blocked or too diversified by a long tabu tenure. Shorter observation period ϕ is more strict because low fluctuation or low values of fitness may occur more likely in shorter periods of time.

Finally, even when the search does not seem to be trapped or too diversified, mutation may be also performed with small probability $p_{mut} = 0.01$. This component is used due to the fact that even the space of parameters seems to have its local optima, when the fitness function may not seem to indicate problems during the search, but the algorithm is cycling or denying high-quality colorings due to overtraining of the model. Mutation helps to give the algorithm chance to recover from such situations. In this case, the parameter is chosen randomly, too. The "direction" of mutation - incrementing or decrementing is also chosen randomly.

6. EXPERIMENTAL RESULTS

In our experiments, we focused on three types of DIMACS instances, commonly used in current graph coloring literature and often referred to as difficult DIMACS instances [10].

Leighton graphs consist of several cliques, connected by randomly generated edges. Maximum size of a clique, embedded in a Leighton graph, is denoted by a predefined chromatic number of the graph. Leighton graphs represent an abstraction of large scheduling problems [12]. We have chosen four most difficult Leighton graphs, two 15-colorable and 25-colorable instances.

Flat graphs are generated by first partitioning the vertex set into approximately equally sized clusters. Then, edges are put only between vertices in different clusters. Thus, solution of a flat graph represents restoration of the initial clustering. Leighton and flat graphs represent structured difficult instances of the problem [10].

Finally, dsjc graphs are entirely *random graphs*. They were generated with predefined number of vertices and edge density. Contrary to the structured instances, random graphs do not have guaranteed chromatic number [6].

6.1 Experimental results of MEA

Table 1: Borderline intervals for parameter values and their mutations

	Borderlines		Mutation	
parameter	min	max	min	max
α_T	0	10	0	1
A	0	20	0	2
ϕ	500	5000	0	500
b	0	10	0	1
c	0	10	0	1

After a series of preliminary experiments, we have chosen a generic configuration of our algorithm. During investigation of some subset of parameters, the other ones were constant, according to this generic configuration. Table 2 contains the generic values, $|P_0| = |L_e|$ is the size of initial population and the elite list, q and T_b are the maximal lifespan and birth period, r is the reward / punishment for the evaluation, p is the percentage of precise agents and β influences the local search length. During the investigation, we used time limit of 10 minutes per each run of the algorithm.

In preliminary experiments, we observed that lazy approach is efficient only on Leighton graphs $le450\_15c$ and $le450\_15d$. These graphs generate extremely large plateaux that can be evaded this way. For other instances, we used a population of precise agents ($p = 100\%$). Parameter $|P_0| = |L_e|$ influences the number of colorings that are explicitly stored. Optimal value strongly depends on instance. For example, for easier instances like $le450\_15x$ or $dsjc500.1$, it is more efficient to use smaller values, because larger population and elite list would only make the algorithm slower. On the other hand, for instances like $le450\_25x$ or $dsjc500.9$, where single TabuCol is not efficient, larger elite list is needed. We finally chose $|P_0| = |L_e| = 20$ that was shown to be a good compromise.

Table 3 illustrates influence of β on performance of MEA. The first three columns contain the instance name, the number of vertices and the chromatic number and the number of colors used by the algorithm. The following two columns contain parameter values for p and β. The next columns contain the success rate (number of successful runs / number of all runs), average number of colorings evaluated during the search (in millions), average number of local search iterations (in thousands) and average running time of MEA on one run of the algorithm. This structure of "scoresheet" is used also in the following tables. On $le450\_15c$, we observed that short tabu search subroutines generate more changes in the neighborhood size $|V^*|$ and provide more dynamics to the lazy approach we used. Due to this fact, lazy population works well with smaller values of β. However, for a more typical instance $dsjc500.1$, we observed that small values of β are destructive, because the local search simply does not have enough time to overcome local optima and the elite list either changes very slowly or is not able to change at all. For precise approach, we finally chose $\beta = 100$. On $dsjc500.1$, this seems not to be the optimal choice, but it is important to provide enough time also to more tricky instances.

The maximal lifespan q and period of birth T_b are the parameters of natality and mortality in MEA. After some time, the population size stabilizes its value close to $\frac{q}{T_b}$. This may be changed by the evaluation operation, when lifespan is modified, and influenced by the instance, however, the mentioned formula is a good estimate. Table 4 contains results regarding these parameters. First, we can see that a short life (small q) has a similar influence as smaller β. The difference between these scenarios, however, is that by using a small β, the milestones (see Section 4) are more dense than by simply a shorter lifetime of an agent. Parameters q and T_b work well in configurations, when the population is larger in the beginning, to provide diversity, and smaller after some time, to provide speed for the algorithm. This is why we finally used $q = 500$ and $T_b = 100$.

Table 5 contains detailed results of MEA on the DIMACS instances. The structure is similar to previous tables. Variant / limit column contains information about the parameters we used. For lazy variant, we used $\beta = 5$ and $p = 0\%$, for precise variant, we used $\beta = 100$ and $p = 100\%$. The time limit was either 1 hour or 3 hours.

As we have mentioned, on $le450\_15x$ Leighton graphs, lazy variant of MEA performs very smoothly. However, precise approach does not achieve such results at all. Surprisingly, on $le450\_25x$ instances, the situation is reversed. We have obtained much better results by using a pure population of precise agents. Regarding success rates and time limit, performance of MEA on Leighton graphs is very encouraging. On the flat graphs, MEA performs comparably to TabuCol. This is caused by the fact that flat graphs are very specific instances, for which a partial representation [1] or a learning-based approach such as PRTS seems to be generally more appropriate. On random graphs, MEA achieves more interesting results. On $dsjc500.5$ instance, it is able to stabilize results of TabuCol to provide a solid success rate with 49 colors and even manages to find a 48-coloring. On $dsjc500.9$ instance, it is able to find 126-colorings, while for TabuCol, current literature reports only 127-colorings [1, 9].

Based on these observations, we can say that MEA clearly outperforms TabuCol. Comparison to other algorithms will be provided in the next section.

6.2 Experimental results of PRTS

Contrary to MEA, PRTS does not need parameter investigation to be properly set. Intervals for each parameter can be determined quite intuitively and p_{mut} is just set to a small value, in our case 0.01 (see Section 5). No other information is needed. Because PRTS is an extension of TabuCol, we are interested in comparison between these two algorithms that is presented in Table 6. The time limit was 1 hour for both algorithms.

On $le450\_15x$ instances, PRTS definitely outperforms TabuCol. Large plateaux that were evaded by lazy approach in MEA, are now overcome by the ability of PRTS to learn the right tabu tenure parameters. On $le450\_25x$ instances, PRTS is much slower, due to the learning effort. No 25-

Table 2: Generic configuration of MEA

$\|P_0\| = \|L_e\|$	q	T_b	r	p	β
20	500	100	1	100%	100

Table 3: Influence of local search length (β) on MEA

G	$\|V\|, \chi$	c	p	β	succ.	st. $\times 10^6$	it. $\times 10^3$	CPU
$le450\_15c$	$450, 15$	15	0%	5	5/5	662	2370	84 s
				20	5/5	2881	14791	6 m
				50	5/5	1255	4935	3 m
				100	4/5	2140	9626	4 m
$dsjc500.1$	$500, ?$	12	100%	5	1/5	10855	96586	8 m
				20	5/5	5606	38757	3 m
				50	4/5	9616	68326	5 m
				100	4/5	9582	64545	5 m

colorings were obtained. However, embedding PRTS in a population-based method or using simply a longer time limit would be sufficient ways how to find these colorings. Flat graphs provide very interesting results. On $flat\_300\_26\_0$, PRTS is surprisingly much faster than TabuCol. On graph $flat\_300\_28\_0$, PRTS was even able to find a 30-coloring, which is known to be hard, even MEA did not manage to find it, along with many other population-based algorithms. Although the success rate is low, we have confirmed this achievement by reobtaining 30-colorings also in repeated experiments. These results indicate overtraining of parameters in the standard dynamic tabu tenure. Note that landscape of the flat graphs' state space is quite specific, for example, on $flat\_300\_28\_0$, algorithms either find a 30-coloring with more $(3-5)$ collisions or directly an optimal solution. However, on $flat\_1000\_76\_0$, PRTS does perform worse than TabuCol, so there does not seem to be a clear connection between topology and tabu tenure parameters. On random graphs, performance of PRTS is also varied. While on $dsjc500.1$, PRTS is slightly slower and on $dsjc500.9$ it does achieve worse success rate, on $dsjc500.5$, PRTS performs much more reliably than TabuCol. This performance indicates that PRTS is also useful, when TabuCol is not able to achieve high success rate due to the overtraining issues.

Summarizing these results and regarding the numbers of colors that algorithms need, we can say that PRTS outperforms TabuCol. There are instances, where PRTS is slower, however, it does not clearly increase the number of colors needed to find an optimal coloring.

7. CONCLUSION AND DISCUSSION

At this point, we summarize results of MEA and PRTS and provide a high-level comparison of their results to 3 other local search methods and 4 other population-based algorithms. Table 7 summarizes the numbers of colors successfully used by all 9 algorithms.

Between the population-based algorithms, only MEA and DCNS [13] do not use crossovers. Thus, a comparison between these algorithms might be interesting. Although MEA and DCNS use different representations, they perform comparably on structured graphs. However, on random graphs, MEA achieves better results. Other current research works also provide evidence that representation used by MEA is more appropriate for random graphs [1]. When comparing algorithms with crossovers and without crossovers, we can find some quite interesting observations. While algorithms with crossovers clearly perform better on random graphs, Leighton graphs do not seem to need a crossover at all. On the flat graphs, situation is even more extreme. While on $flat300\_28\_0$, local search clearly outperforms population-based algorithms, on $flat1000\_76\_0$, crossovers improve results quite significantly.

Regarding the results of PRTS, one should compare them especially to other local search algorithms. Here, PRTS outperforms TabuCol and provides competitive results to PartialCol. The only method that provides stronger results than PRTS, is the Variable Space Search (VSS) [9], which is a much more complicated algorithm that uses 3 different representations and state spaces. Thus, we can say that PRTS is a very good compromise between implementation complexity and performance.

This analysis shows that, indeed, different instances need different approaches. Even when there are algorithms that are able to provide strong performance on a majority of instances [11, 14], they generally synthesize crossovers, local search and diversification. Thus, these methods are not trivial to implement. On the other hand, MEA and PRTS achieve remarkable results by using only intuitive and quite general mechanisms. Finally, we believe that ideas such as using of lifespans or learning-based tabu scheme may open possibilities to find new strong heuristics not only for the graph coloring problem but also to other combinatorial optimization problems.

Acknowledgement

This contribution was supported by Grant Agency VEGA SR under the grant 1/0141/10.

8. REFERENCES

[1] 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.

[2] D. Brélaz. New methods to color vertices of a graph. *Communications of the ACM*, 22:251–256, 1979.

Table 4: Influence of natality and mortality (q and T_b) on MEA

G	$\|V\|, \chi$	c	p	q	T_b	succ.	st. $\times 10^6$	it. $\times 10^3$	CPU
$le450\_15c$	$450, 15$	15	0%	50	10	5/5	1643	7766	4 m
				500	100	4/5	2140	9626	4 m
				200	10	4/5	2438	11868	5 m
				500	25	4/5	2513	11445	6 m
$dsjc500.1$	$500, ?$	12	100%	50	10	1/5	13242	118500	8 m
				500	100	4/5	9582	64545	5 m
				200	10	3/5	14336	115649	8 m
				500	25	3/5	7321	4609	6 m

Table 5: Computational results of MEA

G	$\|V\|, \chi$	c	variant / limit	succ.	st. $\times 10^6$	it. $\times 10^3$	CPU
$le450\_15c$	$450, 15$	**15**	lazy / 1 h	10/10	1221	5554	4 m
$le450\_15d$	$450, 15$	**15**	lazy / 1 h	10/10	2521	13911	7 m
$le450\_25c$	$450, 25$	**25**	precise / 1 h	7/10	63434	186748	33 m
		26		10/10	259	440	5 s
$le450\_25d$	$450, 25$	**25**	precise / 1 h	8/10	91366	214872	42 m
		26		10/10	2969	562	5 s
$flat300\_26\_0$	$300, 26$	**26**	precise / 1 h	10/10	4960	1342	80 s
$flat300\_28\_0$	$300, 28$	31	precise / 1 h	10/10	11172	15466	3 m
$flat1000\_76\_0$	$1000, 76$	87	precise / 3 h	1/5	510717	191261	3 h
		88		5/5	103981	45848	39 m
$dsjc500.1$	$500, ?$	**12**	precise / 1 h	10/10	7756	47880	4 m
$dsjc500.5$	$500, ?$	**48**	precise / 3 h	1/5	583309	270404	3 h
		49		5/5	117298	83177	40 m
$dsjc500.9$	$500, ?$	**126**	precise / 3 h	3/5	368142	174580	2 h

Table 6: Computational results of PRTS and TabuCol within time limit of 1 hour

Instance			Pseudo-reactive TS				TabuCol (basic TS)			
G	$\|V\|, \chi$	c	succ.	st. $\times 10^6$	it. $\times 10^3$	CPU	succ.	st. $\times 10^6$	it. $\times 10^3$	CPU
$le450\_15c$	$450, 15$	**15**	10/10	6606	19807	4 m	0/10			
		16	10/10	660	606	20 s	10/10	54	242	3 s
$le450\_15d$	$450, 15$	**15**	7/10	33316	143418	27 m	0/10			
		16	10/10	890	1115	28 s	10/10	129	781	4 s
$le450\_25c$	$450, 25$	26	10/10	1894	3624	70 s	10/10	33	110	2 s
$le450\_25d$	$450, 25$	26	10/10	1872	3259	64 s	10/10	27	77	1 s
$flat\_300\_26\_0$	$300, 26$	**26**	10/10	651	196	17 s	10/10	2226	614	63 s
$flat\_300\_28\_0$	$300, 28$	30	1/10	93873	140164	58 m	0/10			
		31	10/10	10490	20040	7 m	10/10	10236	14562	7 m
$flat\_1000\_76\_0$	$1000, 76$	88	1/5	83761	48700	51 m	4/5	53696	28470	34 m
$dsjc500.1$	$500, ?$	**12**	10/10	5523	36348	5 m	10/10	3368	26364	3 m
$dsjc500.5$	$500, ?$	49	6/10	51392	56662	35 m	3/10	91285	62784	49 m
$dsjc500.9$	$500, ?$	127	9/10	33119	13707	18 m	10/10	8568	5130	6 m

Table 7: Numbers of colors needed by MEA, PRTS and state-of-the-art algorithms

	Local search algorithms				Population-based algorithms				
G	**PRTS**	TabuCol	PartialCol [1]	VSS [9]	**MEA**	DCNS [13]	GH [7]	EvoCol [14]	MACol [11]
*le*450_15*c*	**15**	16	**15**	**15**	**15**	**15**	**15**	-	**15**
*le*450_15*d*	**15**	16	**15**	**15**	**15**	**15**	-	-	**15**
*le*450_25*c*	26	26	27	26	**25**	**25**	26	**25**	**25**
*le*450_25*d*	26	26	27	26	**25**	**25**	-	**25**	**25**
*flat*300_26_0	**26**	**26**	**26**	**26**	**26**	**26**	-	-	**26**
*flat*300_28_0	30	31	**28**	29	31	31	31	31	29
*flat*1000_76_0	88	88	88	87	87	89	83	**82**	**82**
*dsjc*500.1	**12**	**12**	**12**	**12**	**12**	**12**	-	**12**	**12**
*dsjc*500.5	49	49	49	**48**	**48**	49	**48**	**48**	**48**
*dsjc*500.9	127	127	127	**126**	**126**	128	-	**126**	**126**

[3] F. Comellas and R. Gallegos. Angels & Mortals: A New Combinatorial Optimization Algorithm. *Studies in Fuzziness and Soft Computing*, 166:397–405, 2005.

[4] F. Comellas and J. Martinez-Navarro. Bumblebees: A Multiagent Combinatorial Optimization Algorithm Inspired by Social Insect Behaviour. In *Proceedings of the first ACM/SIGEVO Summit on Genetic and Evolutionary Computation*, pages 811–814. ACM/SIGEVO, 2009.

[5] R. Dorne and J. K. Hao. A new genetic local search algorithm for graph coloring. 1498:745–754, 1998.

[6] e. a. D.S. Johnson. Optimization by simulated annealing: an experimental evaluation; part ii, graph coloring and number partitioning.

[7] P. Galinier and J. K. Hao. Hybrid Evolutionary Algorithms for Graph Coloring. *Journal of Combinatorial Optimization*, 3:379–397, 1999.

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

[9] A. Hertz, M. Plumettaz, and N. Zufferey. Variable space search for graph coloring.

[10] D. Johnson and M. Trick. *Cliques, Coloring, and Satisfiability: Second DIMACS Implementation Challenge*. American Mathematical Society, 1996.

[11] Z. Lü and J. Hao. A Memetic Algorithm for Graph Coloring. *European Journal of Operational Research*, 203(1):241–250, 2010.

[12] F. Leighton. A graph coloring algorithm for large scheduling problems.

[13] C. Morgenstern. Distributed coloration neighborhood search. *Cliques, Coloring, and Satisfiability: Second DIMACS Implementation Challenge*, pages 335–358, 1996.

[14] D. Porumbel, J. Hao, and P. Kuntz. Diversity Control and Multi-Parent Recombination for Evolutionary Graph Coloring Algorithms. In *Proceedings of the 9th European Conference on Evolutionary Computation in Combinatorial Optimization*, pages 121–132, 2009.