Heuristic Methods for Graph Coloring Problems

A. Lim, Y. Zhu
Department of IEEM
Hong Kong Univ of Sci & Tech
Clear Water Bay
Kowloon, Hong Kong
{iealim,zhuyi}@ust.hk

Q. Lou
Department of CS
National Univ of Singapore
3 Science Drive 2
Singapore
louqin@comp.nus.edu.sg

B. Rodrigues
School of Business
Singapore Management Univ
469 Bukit Timah Road
Singapore
br@smu.edu.sg

ABSTRACT

In this work, the Graph Coloring Problem and its generalizations - the Bandwidth Coloring Problem, the Multicoloring Problem and the Bandwidth Multicoloring Problem - are studied. A Squeaky Wheel Optimization with Tabu Search heuristic is developed and experiments using benchmark geometric test cases show that the algorithm performs well for these problems and achieves results for the Bandwidth Multicoloring Problem which improve on results obtained by other researchers.

Categories and Subject Descriptors

F.2.2 [**Nonnumerical Algorithms and Problems**]: [Sequencing and scheduling]

Keywords

Heuristics, Optimization, Tabu Search, Graph Coloring

1. INTRODUCTION

The Graph Coloring Problem (GCP) is a well-known **NP**-complete problem that has been studied extensively. Heuristics have been widely used for the GCP (see, for example, [1]); these include: iterative Greedy by Culberson and Luo [3], Tabu Search by Hertz and de Werra [6] and Glover et al. [5], DSATUR by Brelaz [1], XRLF by Johnson et al [7]. Other techniques, such as combining of the S-IMPASSE algorithm with exhaustive search [10] and a distributed coloration neighborhood search [13] have also been introduced. Among these, the well-known Greedy method is the simplest which takes an ordering of nodes of a graph and colors these with the smallest color satisfying the constraints that no adjacent nodes are assigned same colors. However, the Greedy method performs poorly in practice. DSATUR uses a heuristic which changes the ordering of nodes and then uses the Greedy method to color these nodes. Further to this, exact algorithms, such as branch and bound, have been used to solve instances with a small number of colors [16]. Most of the early work done on this problem can be found in: “Cliques, Coloring and Satisfiability, Second DIMACS Implementation Challenge” [8].

The Multi-Coloring Problem (with weighted nodes), the Bandwidth Coloring Problem (with weighted edges) and the Bandwidth Multi-Coloring Problem (with weighted edges and nodes) are generalizations of the GCP and have a range of applications. The Multi-Coloring Problem can be used, for example, to schedule jobs with different time requirements where the set of colors assigned to a node corresponds to resources (e.g. men) assigned to a job. Here, each node has a demand of a set of colors which we have to assign to it ensuring that its neighbors receive disjoint sets (e.g. sets of men). A related application is in scheduling processor tasks where certain resources cannot be shared concurrently by certain sets of jobs. The problem of channel frequency assignment arises with different transmitters operating in the same region which can interfere with each other. This is common to mobile or general radio networks. The problem can be viewed as a multicoloring problem where each node of an interference graph has an associated integer weight representing the number of calls that must be served by the corresponding base station. We then seek to assign the weighted number of colors to each node such that adjacent nodes are assigned disjoint sets of colors. The objective is then to minimize the number of colors used for such an assignment. Because of signal interference in networks, adjacent transmitters need to operate at different and separated frequencies. This separation requirement is addressed by constraining frequencies to be separated by stipulated ”distances”, usually some positive integer. In such problems, a ”bandwidth” is associated with each frequency.The Bandwidth Coloring and Multi-Coloring Problems have been used for channel frequency assignment with interference problems. In the Fixed Channel Assignment Problem [14], for example, frequency demand is forecasted and used to determine network requirements and frequencies are assigned in advance. In this problem, each cell requires a single assigned frequency and the Bandwidth Coloring Problem can be used. In general, when cells require a number of separated frequencies, the Bandwidth Multi-Coloring Problem is used. For example, this problem can be applied to the Fixed Channel Assignment Problem when co-channel constraints, adjacent channel constraints and the co-site constraints are included [14].

Recently, Prestwich proposed a combination of local search and constraint propagation in a method called FCNS to solve generalized graph coloring problems [15]. This approach is a hybrid of the DSATUR backtracker and the S-IMPASSE local search algorithm and was applied in experiments to geometric graphs. More recently, Lim et al. proposed a hybrid method which combines Squeaky Wheel Optimization (SWO) and hill-climbing techniques which was also tested on geometric graphs [12]. Squeaky Wheel Optimization (SWO) is a meta-heuristic in which the core is a Construct-Analyze-Prioritize cycle [2, 9], which we will introduce in details in Section 3.3. In this work, we propose new algorithms which use Tabu Search and SWO. In particular, the SWO with Tabu Search hybrid allows SWO diversification to be complemented by Tabu Search intensification in search. We found that this approach gives superior results when compared with those obtained by Prestwich and Lim et al. [15, 12] in the Bandwidth Multi-Coloring Problem which is the most difficult of those compared.

This paper is organized as follows. In Section 2, we will give definitions of the GCP and its generalizations - the Bandwidth Coloring Problem, the Multi-Coloring Problem (MCP) and the Bandwidth MCP. In Section 3, we will describe the algorithm, its framework and each of its components - a Greedy Algorithm, SWO and Tabu Search. In Section 4, we will present experimental results with comparisons and analysis. In the last section, we will provide a conclusion.

2. PROBLEM DESCRIPTIONS

We give brief descriptions of the problems we study here.

2.1 The Graph Coloring Problem

Problem: Given a graph $G(V, E)$, find a minimum k, and a mapping $r : V \rightarrow \{1, ..., k\}$ such that $r(i) \neq r(j)$ for each edge $(i, j) \in E$.

This is the classical problem when each node in the graph is assigned one color and colors for adjacent nodes must be different. Many practical applications can be modelled as GCPs, as we mentioned above. This problem has been well-studied. In this problem, each node can only be assigned one color and there are no "distance" restrictions for adjacent nodes. The GCP can be extended as in the following.

2.2 The Bandwidth Coloring Problem

Problem: Given a graph $G(V, E)$ with edge weights $d(i, j)$, find a minimum k and a mapping r from $V \rightarrow \{1, ..., k\}$ such that $r(i) \neq r(j)$ and $|r(i) - r(j)| \geq d(i, j)$ for each edge $(i, j) \in E$.

In this model, the additional constraint that the distance between $r(i)$ and $r(j)$ is at least $d(i, j)$ for each edge can be used to model certain applications, such as channel assignment problem, where edge weights are interpreted as the minimum required "separation distance" between two adjacent stations, cells, etc.

2.3 The Multi-Coloring Problem

Problem: Given a graph $G(V, E)$ with node weights $k(i)$ for each $i \in V$, find a minimum k and subsets $S(i) \subset \{1, ..., k\}$ such that $|S(i)| = k(i)$ for each $i \in V$ and $S(i) \cap S(j) = \emptyset$ for each $(i, j) \in E$ (where $|A|$ denotes the cardinality of set A)

In this model, each node can be assigned multiple colors. Figure 1 is an example of a valid Multi-Coloring instance with $k = 8$.

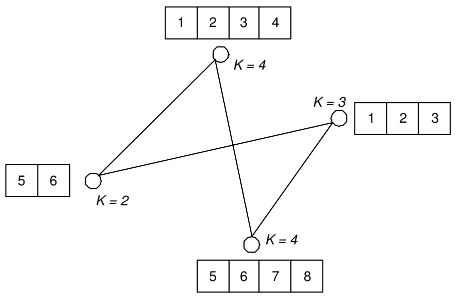

Figure 1: An instance of the Multi-coloring Problem

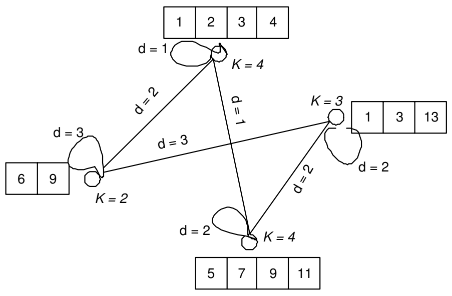

Figure 2: An instance of the Bandwidth Multi-coloring Problem

2.4 The Bandwidth Multi-Coloring Problem

Problem: Given a graph $G(V, E)$ with node weights $k(i)$ for $i \in V$, and edge weights $d(i, j)$ for $(i, j) \in E$, find a minimum k and subsets $S(i) \subset \{1, ..., k\}$ for each $i \in V$, such that $|S(i)| = k(i)$ for each $i \in V$ and such that $S(i) \cap S(j) = \emptyset$ and where, for each $p \in S(i)$ and $q \in S(j)$, $|p - q| \geq d(i, j)$ for each $(i, j) \in E$.

This model combines the above two models, and is more complex. Note that the graph can contain self-loops. For example, $d(1, 1) = 2, k(1) = 3$ means the 3 colors assigned to node 1 must have a difference of value 2 at least between any two of them. The example given in Figure 1 is not a valid bandwidth Multi-Coloring instance since it violates the bandwidth constraint. The example given in Figure 2 is an example of a valid instance of the problem.

3. A HEURISTIC APPROACH

In this section, we discuss a heuristic approach to tackle generalized graph coloring problems. Since the Bandwidth MCP is a generalization of the other three problems, we apply the algorithm only to it.

We will first present the framework of the approach, then discuss its components — Greedy Algorithm, SWO and Tabu Search (TS) respectively.

3.1 Algorithm Framework

In the algorithm, we modeled solutions as sequences of nodes. Given a sequence of "problem elements", in the context of the SWO approach, we used a greedy algorithm to assign colors to it for a solution and then apply meta-heuristics to adjust these sequences to achieve good or better solutions.

Here, the nodes in the graph are the natural candidates to

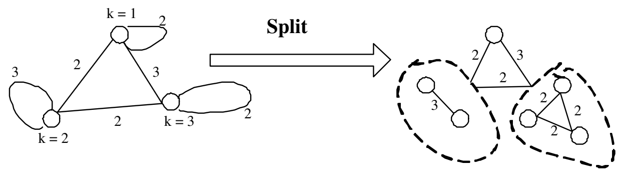

Figure 3: A Instance for Splitting the Nodes in the Graph

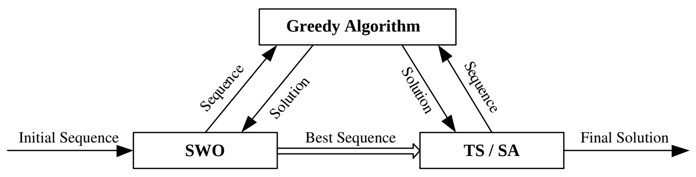

Figure 4: The Framework of the Heuristic Approach

be "problem elements" for the GCP. However, in the Bandwidth MCP, this is not easily applied because each node will be assigned several different colors, between which there are constraints. To deal with this, we split a node with weight k into k nodes (except $k = 1$), and treated each new node as the "problem element" in the context of SWO. Hence, node i with weight k becomes a $k-clique$ (a complete subgraph) in the new graph, with each edge having weight $d(i, i)$ which was originally the loop edge weight. This process is illustrated in Figure 3. If the original graph had n nodes and node i had weight $k(i), 1 \leq i \leq n$, then the new graph will have $\sum_{i=1}^{n} k(i)$ nodes.

When a sequence is given, it will be passed to the greedy algorithm to generate a solution. The greedy algorithm is "deterministic" or one-one correspondence, i.e., for two identical sequences, resulting solutions are the same. This allowed us to exploit meta-heuristic to adjust sequences to find good or better solutions.

The algorithm can be divided into two parts. In the first part, the SWO technique is applied to adjust sequences globally, in particular, through diversification. From this, the best solution that SWO finds is passed to a TS heuristic for refinement and better solutions. We adopted TS as the second meta-heuristics here because of its strength in searching for good solutions in a local search space while avoiding local optima.

The greedy algorithm will then be invoked to evaluate the sequence so as to determine the number of colors it requires. The entire framework is summarized in Figure 4.

3.2 The Greedy Algorithm

The greedy algorithm takes a sequence of "split nodes" and assign colors to them greedily. The idea is straightforward. Following the node sequence, we assign colors one by one where, for each node, we assign the smallest possible color not used by adjacent earlier nodes. The details of this algorithm is shown as Algorithm 1. Here,

n is the number of nodes in the original graph;

$d(i, j)$ is the weight of edge (i, j);

m is the number of nodes after splitting;

$p[1..m]$ is the priority sequence for the m nodes ($p[1] = u$ implies node u has the highest priority);

$c[1..m]$ records the assigned colors.

and the function get_original_node(u) returns the node v in the original graph such that u is split from v.

Algorithm 1 The Greedy Algorithm which Assigns Colors

```
for $i = 1$ to $m$ do
  $c[i] \leftarrow -1$
end for
for $i = 1$ to $m$ do
  $forbidden\_set \leftarrow \emptyset$
  $u \leftarrow p[i]$
  $v \leftarrow$ get_original_node($u$)
  for each node $s$ that adjacent to $u$ do
    if $c[s]! = -1$ then
      $t \leftarrow$ get_original_node($s$)
      $a \leftarrow Max\{0, c[s] - d(v, t) + 1\}$
      $b \leftarrow c[s] + d(v, t) - 1$
      $forbidden\_set \leftarrow forbidden\_set \cup [a..b]$
    end if
  end for
  $c[u] \leftarrow Min\{r, r \in N, r \notin forbidden\_set\}$
end for
```

3.3 Squeaky Wheel Optimization

"Squeaky Wheel" Optimization (SWO) is a meta-heuristic in which the core is a Construct-Analyze-Prioritize cycle [2, 9]. A solution is constructed by a **constructor** using a greedy algorithm where decisions are determined by the priorities assigned to the elements of the problem. The **analyzer** then takes over and is the component that looks for those elements that are "trouble makers" and assigns numerical "blame" values to these problem elements i.e. those elements that contribute to weaknesses in the current solution. In [9], blames are assigned to the nodes that use new colors, for example. A **prioritizer** will then modify the sequence of problem elements and increase the priorities of the trouble makers which then causes the constructor to process them earlier in the next iteration. The cycle repeats until a specified termination condition (number of iterations) is satisfied.

The SWO technique has been successful in applications to Machine Scheduling and Graph Coloring Problems [2, 9]. Here we used SWO to generate an initial solution. One reason for this choice is that SWO is fast when compared with other meta-heuristics such as TS, Simulated Annealing and Genetic Algorithms. The other reason is that SWO is able explore the search space more widely and can provide for good diversification in search. It considers both the solution space and the priority space where small changes in the priority space can cause a large change in solution space. This property causes SWO to view the solution space more globally. The components of SWO, the **constructor**, the **analyzer** and the **prioritizer**, are described as follows:

- **The Constructor:** A greedy algorithm is invoked. The priority sequence of the "problem elements" - the split nodes - is passed to the algorithm and a corresponding greedy solution constructed.

- **The Analyzer:** The analyzer will identify the so-called "trouble makers" and assign blame values to them.

- **The Prioritizer:** Once blame has been assigned, the prioritizer modifies the previous sequence of problem elements. The nodes with higher blame values will be moved forward in the priority sequence while the ones with small blame values will remain in the back of the sequence. This causes the "trouble makers" to be handled first by the constructor in the next iteration.

Here, we used a new scheme to calculate the blame values for the implementation of SWO. Assume that, in the current solution, the maximum color is K. We define "trouble makers" as those elements that use colors which are "large enough". If the node is assigned color c, then c is "large enough" if $c > \alpha.K$, where α is a real number (called a blame rate) in $(0, 1)$ and close to 1. Problem elements will be assigned a constant blame value b.

The SWO algorithm is described in Algorithm 2.

Algorithm 2 The SWO procedure

```
Initialize a random priority sequence $pri$
$max\_value \leftarrow \infty$
**for** $i = 1$ to maximum SWO iterations **do**
  invoke the greedy algorithm with the sequence $pri$
  $cur\_value \leftarrow$ the maximum color in the current solution
  **if** $cur\_value < max\_value$ **then**
    $max\_value \leftarrow cur\_value$
    update best solution
  **end if**
  **for** $j$=1 to $m$ **do**
    $blame[j] \leftarrow j$
    **if** $c[j] > \alpha * cur\_value$ **then**
      $blame[j] \leftarrow blame[j] + b$
    **end if**
  **end for**
  sort the sequence $pri$ in ascending order of blame values
**end for**
```

We first ran the SWO for a number of iterations and picked up the best solution to be the initial solution of TS, which we discuss in the next section.

3.4 Tabu Search

TS is iterative method designed to help standard local search methods escape local optima operating through neighborhood moves that proceed from one solution to another at each iteration. In basic TS, some moves are *tabu* and forbidden unless they lead to desirable outcomes [4].

As mentioned earlier, we only considered sequences of problem elements (split nodes). The neighborhood move is an exchange of two nodes in the solution sequence. The move can be denoted $a \leftrightarrow b$ for $1 \leq a, b \leq m$, where m is the number of nodes in the graph after splitting. A *tabu memory* is used to forbid a reverse move, within a certain number of iterations, to avoid traps at local optima and the number of iterations for which a particular restriction remains is given by a *tabu tenure*. Taking *iter* to denote the current iteration, after we select the move $a \leftrightarrow b$ as the best one from a number of neighborhood moves (specified as No. TS moves in the next section), *tabu memory* is updated as *tabu* $(a, b) = iter + tabu\ tenure$. The reverse move $b \leftrightarrow a$ is then prevented in the next *tabu tenure* number of iterations.

The TS procedure is detailed in Algorithm 3.

Algorithm 3 The Tabu Search Procedure

```
get initial solution $x_{now}$ from SWO method
$x_{best} \leftarrow x_{now}$
$iter \leftarrow 0$
**while** $iter \leq max\_iter$ **do**
  generate the neighborhood set $N(x_{now})$ by the exchange move
  evaluate each candidate solution $x_{trial}$ as $f(x_{trial})$, by the greedy algorithm
  $x_{next} \leftarrow \emptyset$
  **for** all $x_{trial}$ exchanging elements $a$ and $b$ **do**
    **if** $iter > tabu(a \leftrightarrow b)$ and $f(x_{next}) > f(x_{trial})$ **then**
      $x_{next} \leftarrow x_{trial}$
      $p \leftarrow a, q \leftarrow b$
    **end if**
  **end for**
  $tabu(p \leftrightarrow q) \leftarrow iter + tabu\_tenure$
  $x_{now} \leftarrow x_{next}$
  **if** $f(x_{best}) > f(x_{now})$ **then**
    $x_{best} \leftarrow x_{now}$
  **end if**
  $iter \leftarrow iter + 1$
**end while**
```

4. EXPERIMENTAL RESULTS

To test the algorithms, we used the 33 benchmark geometric instances given by Trick which can be found in Computational Symposium Announcement: Graph Coloring and its Generalizations at *http://mat.gsia.cmu.edu/COLORING02/*. Points are generated in a 10,000 by 10,000 grid and are connected by an edge if they are close enough together. Edge weights are inversely proportional to the distance between nodes and node weights are uniformly generated. *GEOMn* instances are sparse; *GEOMa* and *GEOMb* instances are denser and require fewer colors per node.

We implemented the SWO+TS algorithm using Java and with the results, we list those from [15] and [11], for experiments on the same set of instances.

Table 1 gives the parameter settings for the four problems. We experimented with different number of restarts and found that after 10 restarts the marginal improvement was small (with a maximum of 2 colors in only two cases out of 33), and beyond 15 restarts, the improvement was negligible. With more restarts, however, running times increased. We therefore chose to restart the algorithm 10 times with different (randomized) initial solutions. The comparisons between the method and the previous two methods is shown in Table 2 where the following notations are used:

- M1 - Method used by Prestwich [15] [1]
- M2 - Method used by Lim et al. [11] [2]
- M3 - The SWO+TS method [3]

Table 3 and Table 4 give detailed results for the MCP and Bandwidth MCP (The detailed results of GCP and Band-

[1]The benchmark program dfmax(r500.5.b) running time: 35.21 seconds (user time).
[2]The benchmark program dfmax(r500.5.b) running time: 25.32 seconds (user time).
[3]The benchmark program dfmax(r500.5.b) running time: 74.12 seconds (user time).

Parameters	Pure Coloring	Bandwidth Coloring	MCP	Bandwidth MCP
Restarts No.	10	10	10	10
SWO Iteration	200	400	400	400
Blame Rate α	0.95	0.95	0.95	0.95
Blame Value b	4	4	4	8
TS Iteration	100	100	200	200
No. of TS Moves	40	40	50	50
TS Tenure	5	5	5	5

Table 1: Parameter Settings

	Pure Coloring		Bandwidth Coloring		MCP		Bandwidth MCP	
	M1	M2	M1	M2	M1[1]	M2	M1[2]	M2
Wins	0	0	0	17	-	0	16	22
Ties	30	31	15	14	-	33	1	11
Loses	3	2	18	2	-	0	0	0
Max Wins	0	0	0	5	-	0	77	11
Avg. Wins	0	0	0	1.53	-	0	20.75	5.23
Max Loses	1	1	11	2	-	0	0	0
Avg. Loses	1.00	1.00	4.61	1.50	-	0	0	0

[1] Not presented in [15]

[2] Only 17 instances presented in [15]

Table 2: Comparison of M3 with M1 and M2

width Coloring are not presented here due to the space limitation), where k denotes the number of colors and t denotes the running time in seconds. In [15], when there are no results for the MCP, and only partial results for the Bandwidth MCP, we write "-" in the tables to denote this.

The performance of the algorithms is good in all four problems and especially so for the Bandwidth MCP. We conducted the experiments using Bandwidth MCP instances which are not covered in [15]. Results tied or surpassed all of the results in [15] and [11]. Although the methods that are applied in [15] are good for GCP and perform well on the DIMACS geometric graph instances, the new algorithm is only weaker in 3 out of 33 cases and for only 1 color each. In the Bandwidth Coloring Problem, results are not as good as [15] although comparable with [11].

We believe that the method in [15] does not work well for the Bandwidth MCP because there are too many variables and constraints to check simultaneously. The backtracker used there would take a long time to deal with the many constraints. However, the approach given in [11] and here always generates a solution quickly and then improves on it incrementally. One of the reasons why the method outperforms [11] is that TS works well with SWO and is able to avoid local optima. Furthermore, we used TS after SWO, whereas hill-climbing was implemented in each iteration of the SWO in [11] which results in longer times. In the same amount of time, we were able to run more iterations of SWO to refine the solutions.

It is interesting that in the MCP, we obtained the same results as [11] on all the 33 instances. No results are provided in [15] for this problem.

In terms of running time, the method takes relatively longer times for small instances, especially for the Coloring and Bandwidth Coloring. However, the running times did not increase sharply when instances became larger, and are comparable with the other two methods. We believe this is due to the different heuristics setup. Since we used constant parameters, running times will not change much as the sizes vary; however, the running times of the other heuristics were dependant on the instance sizes.

5. CONCLUSION

In this work, we discussed several variations of generalized graph coloring problems and proposed a new hybrid SWO with TS technique to solve these. Experiments are conducted using benchmark problems and results compared with those obtained by existing methods. These results show that this approach is effective for the problems studied and provides superior results than the current methods for the Bandwidth MCP.

6. REFERENCES

[1] D. Brelaz. New methods to color the vertices of a graph. *Communications of ACM*, 22(4):251–256, 1979.

[2] David P. Clements, James M. Crawford, David E. Joslin, Geoge L. Nemhauser, Markus E. Puttlitz, and Martin W.P. Savelsbergh. Heurstic optimization: A hybrid ai/or approach. In *Proceedings of the Workshop on Industrial Constraint-Directed Scheduling*, 1997.

[3] Joseph C. Culberson and Feng Luo. Exploring the k-colorable landscape with iterative greedy. *Cliques, Coloring and Satisfiability, Second DIMACS Implementation Challenge*, 1993.

[4] Fred Glover and Manuel Laguna. *Tabu Search.* Kluwer Acadamic Publishers, 1997.

[5] Fred Glover, Mark Parker, and Jennifer Ryan. Coloring by tabu branch and bound. *Cliques, Coloring*

and Satisfiability, Second DIMACS Implementation Challenge, 1993.

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

[7] D. S. Johnson, C. A. Aragon, L. A. Mcgeoch, and C. Schevon. Optimization by simulated annealing: An experimental evaluation c part ii (graph coloring and number partitioning). *Operations Research*, 31:378–406, 1991.

[8] David S. Johnson and Michael A. Trick (Editors). *Cliques, Coloring and Satisfiability, Second DIMACS Implementation Challenge*. American Mathematical Society, 1993.

[9] David E. Joslin and David P. Clements. "squeaky wheel" optimization. In *Proceedings of American Association of Artificial Intelligence National Conference 1998*, pages 340–346, 1998.

[10] Gray Lewandowski and Anne Condon. Expreiments with parallel graph coloring heuristics and applications of graph coloring. *Cliques, Coloring and Satisfiability, Second DIMACS Implementation Challenge*, 1993.

[11] Andrew Lim, Brian Rodrigues, and Yi Zhu. Crane scheduling using swo with local search. In *Proceedings of 4th Asia-Pacific Conference on Simulated Evolution and Learning*, 2002.

[12] Andrew Lim, Xingwen Zhang, and Yi Zhu. A hybrid method for the graph coloring problem and its generalizations. In *5th Metaheuristics International Conference (MIC 2003)*, 2002.

[13] Craig Morgenstern. Distributed coloration neighborhood search. *Cliques, Coloring and Satisfiability, Second DIMACS Implementation Challenge*, 1993.

[14] Eun-Jon Park, Yong-Hyuk Kim, and Byung-Ro Moon. Genetic search for fixed channel assignment problem with limited bandwidth. In *Proceedings of Genetic and Evolutionary Computation Conference*, pages 1772–1779, 2002.

[15] Steven Prestwich. Constrained bandwidth multicoloration neighborhoods. In *Proceedings of Computational Symposium on Graph Coloring and its Generalizations*, 2002.

[16] E. C. Sewell. An improved algorithm for exact graph coloring. *Cliques, Coloring and Satisfiability, Second DIMACS Implementation Challenge*, 1993.

Graph	M1		M2		M3		Graph	M1		M2		M3	
	k	*t*	*k*	*t*	*k*	*t*		*k*	*t*	*k*	*t*	*k*	*t*
GEOM20	-	-	28	0	28	4	GEOM80	-	-	63	0	63	551
GEOM20a	-	-	30	0	30	3	GEOM80a	-	-	68	0	68	476
GEOM20b	-	-	8	0	8	7	GEOM80b	-	-	25	0	25	111
GEOM30	-	-	26	0	26	61	GEOM90	-	-	51	0	51	697
GEOM30a	-	-	40	0	40	99	GEOM90a	-	-	65	0	65	625
GEOM30b	-	-	11	0	11	20	GEOM90b	-	-	28	0	28	135
GEOM40	-	-	31	0	31	131	GEOM100	-	-	60	0	60	845
GEOM40a	-	-	46	0	46	138	GEOM100a	-	-	81	0	81	854
GEOM40b	-	-	14	0	14	29	GEOM100b	-	-	30	0	30	156
GEOM50	-	-	35	0	35	227	GEOM110	-	-	62	0	62	1022
GEOM50a	-	-	61	0	61	312	GEOM110a	-	-	91	0	91	1172
GEOM50b	-	-	17	0	17	46	GEOM110b	-	-	37	0	37	211
GEOM60	-	-	36	0	36	275	GEOM120	-	-	64	0	64	1268
GEOM60a	-	-	65	0	65	391	GEOM120a	-	-	93	0	93	2349
GEOM60b	-	-	22	0	22	64	GEOM120b	-	-	34	0	34	489
GEOM70	-	-	44	0	44	374							
GEOM70a	-	-	71	0	71	439							
GEOM70b	-	-	22	0	22	84							

Table 3: Experimental Results for MCP

Graph	M1		M2		M3		Graph	M1		M2		M3	
	k	*t*	*k*	*t*	*k*	*t*		*k*	*t*	*k*	*t*	*k*	*t*
GEOM20	159	56	149	0	149	176	GEOM80	-	-	394	4041	383	2159
GEOM20a	175	145	169	16	169	169	GEOM80a	-	-	379	677	379	2006
GEOM20b	44	25	44	0	44	25	GEOM80b	152	1380	145	3230	141	417
GEOM30	168	178	160	0	160	241	GEOM90	-	-	335	4095	332	2623
GEOM30a	235	64	211	10	209	425	GEOM90a	-	-	382	10404	377	2587
GEOM30b	79	14	77	0	77	69	GEOM90b	-	-	157	648	157	489
GEOM40	189	135	167	3	167	496	GEOM100	-	-	413	631	404	3283
GEOM40a	260	24	214	358	213	573	GEOM100a	-	-	462	172	459	3529
GEOM40b	80	94	76	8	74	108	GEOM100b	-	-	172	4893	170	584
GEOM50	257	96	224	41	224	819	GEOM110	-	-	389	577	383	3893
GEOM50a	395	299	326	96	318	125	GEOM110a	-	-	501	4671	494	4661
GEOM50b	89	94	87	53	87	162	GEOM110b	-	-	210	12	206	716
GEOM60	279	514	258	46	258	1016	GEOM120	-	-	409	1825	402	4317
GEOM60a	-	-	368	3748	358	1717	GEOM120a	-	-	564	5335	556	6688
GEOM60b	128	1012	119	300	116	246	GEOM120b	-	-	201	869	199	1028
GEOM70	310	1019	279	25	273	1458							
GEOM70a	-	-	478	417	469	1987							
GEOM70b	133	766	124	136	121	312							

Table 4: Experimental Results for Bandwidth MCP