A BRANCH-AND-PRICE APPROACH FOR GRAPH MULTI-COLORING

Anuj Mehrotra
Department of Management Science
School of Business Administration
University of Miami
Coral Gables, FL 33124-8237
anuj@miami.edu

Michael A. Trick
Tepper School of Business
Carnegie Mellon University
Pittsburgh, PA 15213-3890
trick@cmu.edu

Abstract We present a branch-and-price framework for solving the graph multi-coloring problem. We propose column generation to implicitly optimize the linear programming relaxation of an independent set formulation (where there is one variable for each independent set in the graph) for graph multi-coloring. This approach, while requiring the solution of a difficult subproblem, is a promising method to obtain good solutions for small to moderate size problems quickly. Some implementation details and initial computational experience are presented.

Keywords: Integer Programming, Coloring, column generation, multi-coloring.

1. INTRODUCTION

The graph multi-coloring problem is a generalization of the well-known graph coloring problem. Given a graph, the (node) coloring problem is to assign a single color to each node such that the colors on adjacent nodes are different. For the multi-coloring problem, each node must be assigned a preset number of colors and no two adjacent nodes may have any colors in common. The objective is to accomplish this using the fewest possible number of colors.

Like the graph coloring problem, the multi-coloring problem can model a number of applications. It is used in scheduling ([7]) where each node represents a job, edges represent jobs that cannot be done simultaneously, and the colors represent time units. Each job requires multiple time units (the required number of colors at the node), and can be scheduled preemptively. The minimum number of colors then represents the makespan of the instance. Multi-colorings also arise in telecommunication channel assignment where the nodes represent transmitters, edges represent interference, and the transmitters send out signals on multiple wavelengths (the colors) [14]. It is due to this application in telecommunications that multi-coloring, as well as generalizations that further restrict feasible colorings, dates back to the 1960s. Aardal et al. ([1] provide an excellent survey on these problems.

The multi-coloring problem can be reduced to graph coloring by replacing each node by a clique of size equal to the required number of colors. Edges are then replaced with complete bipartite graphs between the corresponding cliques. Such a transformation both increases the size of the graph and embeds an unwanted symmetry into the problem. It is therefore useful to develop specialized algorithms that attack the multi-coloring problem directly.

Johnson, Mehrotra, and Trick [9] included the multi-coloring problem in a series of computational challenges, and provide a testbed of sample instances. Prestwich [17] developed a local search algorithm for this form of the multi-coloring problem and compared that approach to a satsifiability-based model. Without lower bounds or exact solutions to simple problems, however, it is difficult to evaluate these heuristic approaches.

We suggest an approach based on an integer programming formulation of the graph multi-coloring problem. This formulation, called the *independent set formulation*, has a variable for each independent set in the graph. In our previous work on graph coloring problems [12], we demonstrated that despite the enormous number of variables in this formulation, it is possible to develop an effective column generation technique for the coloring problem. We used appropriate branching rules and tested our branch-and-price approach on a variety of coloring instances. Encouraged by the effectiveness of such a method for coloring problems, we discuss the extension of such an approach on graph multi-coloring problems. This extension is independently interesting due particularly to the non-binary nature of the variables. Most examples of branch-and-price use binary variables, which results in now-routine branching rules. With non-binary variables, we need to explore new and intriguing approaches to branching.

In Section 2, we develop the independent set formulation of the graph multi-coloring problem and discuss various advantages of the formulation. In Section 3, we summarize the techniques for generating columns in this formulation and outline one method for such generation. In Section 4, we discuss the branching rules that are necessary to be developed for a full branch-and-price method. In Section 5, we describe some initial computational results and conclude with some directions for future exploration.

2. A COLUMN GENERATION MODEL

Let $G = (V, E)$ be an undirected graph on V, the set of vertices, with E being the set of of edges. Let $|V| = n$ and $|E| = m$. Let w_i be an integer weight associated with a node $i \in V$ giving the required number of colors at the node. When $w_i = 1$, for all $i \in V$, then the problem is the usual vertex coloring problem.

A *multi-coloring* of G is an assignment of w_i labels to each vertex i such that the endpoints of any edge do not have any common label. A *minimum multi-coloring* of G is a multi-coloring with the fewest different labels among all possible multi-colorings.

An *independent set*, S of G is a set of vertices $S \subseteq V$ such that there is no edge in E connecting any pair of nodes in S. Clearly in any coloring of G, all vertices with the same label comprise an independent set. A *maximal independent set* is an independent set that is not strictly included in any other independent set.

The problem of finding a minimum multi-coloring in a graph can be formulated in many ways. For instance, letting x_{ik}, $i \in V$, $1 \leq k \leq K$ be a binary variable that is 1 if label k is assigned to vertex i and 0 otherwise, where K represents an upper bound on the number of labels needed to obtain a valid multi-coloring of the graph, the problem can be formulated as follows:

$$
\begin{array}{lrcll}
\mathit{Minimize} & y & & & \\
s.t. & x_{ik} + x_{jk} & \leq & 1 & \forall\, (i,j) \in E,\ \ k = 1, ..., K \\
& \sum_k x_{ik} & = & w_i & \forall\, i \in V \\
& y & \geq & k x_{ik} & \forall\, i \in V,\ \ k = 1, ..., K \\
& x_{ik} & \in & \{0,1\} & \forall\, i \in V,\ \ k = 1, ..., K.
\end{array}
$$

We will refer to this formulation as (VC). While correct, (VC) is difficult to use in practice. One obvious problem is the size of the formulation. Since K can be quite large, the formulation can have up to nK variables and $2Km + n$ constraints. Given the need to enforce integrality, this formulation becomes computationally intractable for all except the smallest of instances. This is especially true because the linear programming relaxation is extremely fractional. To see this, note that even when all $w_i = 1$, the solution, $x_{ik} = 1/K$ for every (i, k) is feasible whenever $K \geq 2$.

A second, less obvious, problem involves the symmetry of the formulation. The variables for each k appear in exactly the same way. This means that it is difficult to enforce integrality in one variable without problems showing up in the other variables. This is because any solution to the linear relaxation has an exponential number (as a function of K) of representations. Therefore, branching to force x_{i1} to take on integral values does little good because it results in another representation of the same fractional solution in which x_{i2} takes on the old value of x_{i1} and vice-versa.

To address this problem, we consider a formulation with far fewer constraints that does not exhibit the same symmetry problems as our first formulation. Let T be the set of all maximal independent sets of G. We create a formulation with binary variables, x_t, for each $t \in T$. In this formulation, $x_t = k$ implies that independent set t will be given k unique labels, while $x_t = 0$ implies that the set does not require a label. The minimum multi-coloring problem is then the following (denoted (IS)):

$$
\begin{array}{lll}
\textit{Minimize} & \displaystyle\sum_{t \in T} x_t & \\
\textit{Subject to} & \displaystyle\sum_{\{t : i \in T\}} x_t \quad \geq \quad w_i & \forall\, i \in V \\
& x_t \;\geq\; 0 \;\textit{and integer} & \forall\, t \in T.
\end{array}
$$

This formulation can also be obtained from the first formulation by using a suitable decomposition scheme as explained in [10] in the context of general mixed integer programs. The formulation (IS) has only one constraint for each vertex, but can have a tremendous number of variables. Note that a feasible solution to (IS) may assign more than the specified number of labels to a vertex since we include only maximual independent sets in the formulation. This can be remedied by using any

correct subset of the assigned multiple labels as the labels for the vertex. The alternative would be to allow non-maximal sets in T and to require equalities in (IS). In view of the ease of correcting the problem versus the great increase in problem size that would result from expanding T, we choose the given formulation.

This formulation exhibits much less symmetry than (VC): vertices are combined into independent sets and forcing a variable to 0 means that the vertcies comprising the corresponding independent set will not receive the same color in the solution. Furthermore, it is easy to show [10] that the bound provided by the linear relaxation of (IS) will be at least as good as the bound provided by the linear relaxation of (VC).

The fact remains, however, that (IS) can have far more variables than can be reasonably handled directly. We resolve this difficulty by using only a subset of the variables and generating more variables as needed. This technique, called *column generation*, is well known for linear programs and has emerged as a viable technique for a number of integer programming problems [5, 12]. The need to generate dual variables (which requires something like linear programming) while still enforcing integrality makes column generation procedures nontrivial for integer programs. The procedures need to be suitably developed and their effectiveness is usually dependent on cleverly exploiting the characteristics of the problem.

The following is a brief overview of the column generation technique in terms of (IS). Begin with a subset $\bar{T}$ of independent sets. Solve the linear relaxation (replace the integrality constraints on x_s with nonnegativity) of (IS) restricted to $t \in \bar{T}$. This gives a feasible solution to the linear relaxation of (IS) and a dual value π_i for each constraint in (IS). Now, determine if it would be useful to expand $\bar{T}$. This is done by solving the following *maximum weighted independent set* problem (MWIS):

$$
\begin{array}{lrcll}
\mathit{Maximize} & \displaystyle\sum_{i \in V} \pi_i z_i & & & \\
\mathit{Subject\ to} & z_i + z_j & \leq & 1 & \forall\, (i,j) \in E \\
& z_i & \in & \{0,1\} & \forall\, i \in V.
\end{array}
$$

If the optimal solution to this problem is more than 1, then the z_i with value 1 correspond to an independent set that should be added to $\bar{T}$. If the optimal value is less than or equal to 1, then there exist no improving independent sets: solving the linear relaxation of (IS) over the current $\bar{T}$ is the same as solving it over T.

This process is repeated until there is no improving independent set. If the resulting solution to the linear relaxation of (IS) has x_t integer for all $t \in \bar{T}$, then that corresponds to an optimal solution to (IS) over T. When some of the x_t are not integer, however, we are faced with the problem of enforcing integrality.

To complete this algorithm, then, we need to do two things. First, since (MWIS) is itself a difficult problem, we must devise techniques to solve it that are sufficiently fast to be able to be used repeatedly. Second, we must find a way of enforcing integrality if the solution to the linear relaxation of (IS) contains fractional values. Standard techniques of enforcing integrality (cutting planes, fixing variables) make it difficult or impossible to generate improving independent sets. We discuss these two problems in the next two sections.

3. SOLVING THE MAXIMUM WEIGHTED INDEPENDENT SET PROBLEM

The maximum weighted independent set problem is a well-studied problem in graph theory and combinatorial optimization. Since a clique is an independent set in the complement of a graph, the literature on the maximum weighted clique is equally relevant. Various solution approaches have been tried, including implicit enumeration [6], integer programming with branch and bound [3, 4], and integer programming with cutting planes [2, 15]. In addition, a number of heuristics have been developed [16] and combined with general heuristic methods such as simulated annealing [8]. In this section, we outline a simple recursive algorithm based on the work of [11] and describe a simple greedy heuristic that can be used to reduce the need for the recursive algorithm.

The basic algorithm for finding a maximum weighted independent set (MWIS) in the graph $G(V, E)$ is based on the following insight. For any subgraph $G_1(V_1, E_1)$ of G, and a vertex $i \in V_1$, the MWIS in G_1 is either the MWIS in G_1 restricted to $V_1/\{i\}$ or it is i together with the MWIS in $\mathrm{AN}(i)$, where $\mathrm{AN}(i)$ is the *anti–neighbor* set of i: the set of all vertices j in V_1 such that $(i, j) \notin E_1$. This insight, first examined in [11] for the unweighted case, leads to the following recursion which can be turned into a full program:

$$\mathrm{MWIS}(V_1 \cup \{k\}) = \max(\mathrm{MWIS}(V_1), \mathrm{MWIS}(\{k\} \cup \mathrm{AN}(k))),$$

where $\mathrm{MWIS}(S)$ represents the maximum weighted independent set in the subgraph of G induced by the set of nodes in S.

While this approach is reasonably effective for graphs that are not too sparse, it can be improved by appropriately ordering the vertices to add to V_1. The following have been shown to be effective in reducing the computational burden of the recursion:

- Begin with V_1 equal to a heuristically found independent set. We use a simple greedy approach to find such a set, with the nodes ordered by node weight.
- Order the remaining vertices in order of degree from lowest to highest, and add them to V_1 in that order. During the final stages of the recursion, it is important to keep the anti–neighbor set small in order to solve the MWIS on as small a graph as possible. Since vertices with high degree have small anti–neighbor sets, those should be saved for the end.
- Use simple bounds to determine if a branch of the recursion can possibly return a MWIS better than the incumbent. For instance, if the total weight of the set examined is less than the incumbent, the incumbent is necessarily better, so it is unnecessary to continue the recursion.
- Use a faster code for smaller problems. It appears that a weighted version of the method of Carraghan and Pardalos [6] is faster for smaller problems. This is particularly the case since it is able to terminate when it is clear that no independent set is available that is better than the incumbent. In our tests, which use relatively small graphs, we use a variant of Carraghan and Pardalos for all except the first level of recursion, which echoes the results of Khoury and Pardalos in the unweighted case.

In the context of our column generation technique, it is not critical that we get the best (highest weight) maximal independent set: it is sufficient to get any set with weight over 1. This suggests that a heuristic approach for finding an improving column may suffice in many cases. It is only when it is necessary to prove that no set exists with weight over 1 (or when the heuristics fail) that it is necessary to resort to the recursion. There are many heuristics for weighted independent sets. The simplest is the greedy heuristic: begin with (one of) the highest weighted vertices and add vertices in nonincreasing order of their weight making certain that the resulting set remains an independent set.

This heuristic, in addition to being simple, is very fast, and seems to work reasonably well. The resulting independent set can either be added directly to (IS) (if it has value over 1) or can be used as a starting point for the recursion.

4. BRANCHING RULE

A difficult part about using column generation for integer programs is the development of branching rules to ensure integrality. Rules that are appropriate for integer programs where the entire set of columns is explicitly available do not work well with restricted integer programs where the columns are generated by implicit techniques.

The fact that the variables in (IS) are general integers, rather than binary variables, makes this issue even more difficult. For binary variables, the Ryan-Foster [18] branching rule is generally effective, but that rule cannot be used for general integer variables. For (single-color per node) graph coloring, given a solution to (IS), the Ryan-Foster rule identifies two nodes i and j, such that there is a fractional independent set that includes both i and j. The branching is then on whether i and j have the same color or different colors. For the purposes of generating improving independent sets, this involves either contracting two nodes into one or adding an edge to the graph, respectively, as developed in [12]. Such changes do not affect the operation of the MWIS algorithm.

For general integers, it is not necessarily the case that there will be a pair of vertices with a fractional number of colors in common. Vanderbeck [19] does show there are sets of nodes V_1 and V_2 such that the x values for all independent sets that contain all nodes in V_1 and no nodes in V_2 is fractional. If we let $S(V_1, V_2)$ represent the currently generated independent sets that contain all of V_1 and none of V_2, this leads to a branching rule with

$$\sum_{s \in S(V_1,V_2)} x_s \leq k$$

in one branch, and

$$\sum_{s \in S(V_1,V_2)} x_s \geq k+1$$

in the other. This can complicate the solving of the subproblem (MWIS) since either case involves adding a constraint to (IS). This constraint leads to a dual value that must be considered in the MWIS subproblem.

This problem can be addressed in one of two ways. Vanderbeck [19] gives an approach where multiple subproblems are solved without modifying the structure of the subproblem (in our case, MWIS). This approach has the advantage of keeping the subproblem algorithm the same, at the expense of requiring the solution of multiple subproblems. Further, this approach has the disadvantage that the branching rule needs to be more complicated than the node-pair rule given by the Ryan-Foster

rule. Instead, the branching constraints need to consist of nested sets of constraints.

The alternative approach is to directly embed the dual values associated with branching constraints into the subproblem. To do this, we will have to modify the solution approach to MWIS to allow costs on arbitrary pairs of sets (V_1, V_2). This dual value is charged for any independent set that contains all of V_1 and none of V_2.

Fortunately, this is a straightforward modification of the implicit enumeration approach in [12], similar to the modification we proposed in the context of solving clustering problems [13] where the costs only appeared on edges between nodes.

The key aspect of our implicit enumeration is that, at each step, the nodes of the graph are divided into three sets: those that will be in the independent set (I), those that are definitely not in the independent set (NI), and those for which their status is unknown (UN). The duals associated with (V_1, V_2) can similarly be assigned one of three states: definitely to be charged (C), definitely not to be charged (NC) and "not yet determined" (UC). For instance, if the current independent set contains a member of V_2 we know that the corresponding dual on (V_1, V_2) will not be charged.

At each stage of the implicit enumeration, we can calculate an upper bound by adding in the duals for all nodes in I, all the positive duals in NI, all duals in C, and all positive duals in UC. The lower bound is the sum of the duals in I and C. We can strengthen the bounds somewhat by taking the dual for any entry in UC containing just one node in UN and moving that dual value to the UN node. This gives a valid recursion for the case of dual values on arbitrary node sets.

5. COMPUTATIONAL DETAILS

Our current implementation focuses on first optimizing the LP relaxation of (IS) via column generation. Then we determine the best integer solution to the restricted (IS) formulation comprising of the columns generated to optimize the LP relaxation at the root node of the branch-and-price tree. Here we provide some implementation details and initial computational results that we have obtained.

5.1 Implementation Issues

We generate a feasible initial multi-coloring using the greedy MWIS heuristic repeatedly until all nodes are colored at least once. This gives us an initial solution to the multi-coloring problem as well as a number of columns to add to our linear program. We then generate columns

to improve the linear program. The following discussion pertains to generation of columns to improve the linear program.

Improving the Linear Program.

Improving Column. As mentioned earlier, any solution to the MWIS with value greater than 1 represents an improving column for the linear program. In our current implementation, we set a target to 3.0 and our MWIS algorithm either returns the first such solution it finds, failing which, it finds the exact solution. We have also experimented with changing this target value to a higher number initially (an approach to find a *good* set of columns as fast as possible) and then decreasing its value later on in the column generation. The effort required to solve some difficult problems can be substantially reduced by suitably altering this target value.

Ordering the Nodes. The order in which the nodes are to be considered can be specified in our MWIS algorithm. We have found that ordering the nodes independently by nonincreasing weights or by nonincreasing degree is not as efficient as ordering them by considering both at the same time. In our experiments we order the nodes in nonincreasing values of the square root of the degree of the node times the weight of the node.

Column Management. Another approach to optimizing the linear program more quickly is to generate several columns rather than a single column [5] at every iteration. For example, one could use improvement algorithms that take existing columns with reduced cost equal to zero and try to construct columns that might improve the linear program. In our experiments, we generated more candidates by determining other independent sets at each iteration such that every node belonged to at least one indpendent set being added.

5.2 Computational Results

In our computational experiments, we use instances drawn from a large number of sources. Our goal is to determine the robustness of the approach. For some of these graphs, the coloring problem has no real interpretation. We use these graphs as examples of structured graphs, rather than just experimenting on random graphs. These graphs come from a large test set at http://*mat.tepper.cmu.edu/COLOR*04.

Currently, we have not implemented the branching scheme. Rather, we use the standardized branching to determine an integer solution from

among the independent sets generated at the root node to optimize the corresponding LP relaxation of the (IS) formulation. Hence our current implementation provides an optimization-based heuristic procedure. We report our results in Tables 1 and 2. The instance name identifies the problem from the test set. The objective values corresponding to the optimal LP relaxation solution and the intger solution obtained by our method are listed under the columns labeled LP, and Heur, respectively. The gap between these two objective values and the computational time in seconds to optimize the linear relaxation and then to determine the integer solution are listed in the next three columns. The column labeled *cons* lists the number of constraints in the corresponding (IS) equal to the number of vertices in the graph. The number of independent sets generated to optimize the LP relaxation is listed under the column labeled *vars*. The computational results reported here are limited to the best integer solution found in at most 1000 seconds using CPLEX default branching scheme on DEC ALPHA workstation. As can be seen from the gap between the LP bound and the corresponding (heuristic) integral solution obtained by our methodology, this branch-and-price framework looks promising for finding optimal multi-coloring solutions for small to moderate size graphs. In Table 1, we report results on geometric graphs with up to 120 nodes. The best integer solution found for these is within 1 of the optimal multi-coloring in the worst case. The cpu time is also reasonable. A similar performance is seen for the random graphs of up to 100 nodes except for R100-1ga where the gap is 2 between the LP bound and the best integer solution found in 1000 seconds. The gaps are higher for some miscellaneous graphs in Table 2.

5.3 Further Research

A full implementation of the branching is necessary to complete the branch-and-price framework proposed here. Based on the initial results, there is hope that the LP bound is strong and one may not need to have a very deep branch-and-price tree to find optimal multi-colorings for many structured graphs. Further exploration will explore the robustness of this framework for general graphs.

It will also be interesting to see the comparison between using this branch-and-price scheme with a branch-and-price scheme that uses modified branching scheme proposed by Vanderbeck [19].

Finally, it will be interesting to see if this framework can be suitably exploited to solve other variations and extensions of coloring problems.

Table 1. Results for Geometric Graphs

Instance	LP	Heur	Gap	cpu-lp	cpu-ip	cons	vars
geom20	28.00	28	0	0	0	20	31
geom20a	30.00	30	0	0	0	20	29
geom20b	8.00	8	0	0	0	20	34
geom30	26.00	26	0	0	0	30	49
geom30a	40.00	40	0	0	0	30	65
geom30b	11.00	11	0	0	0	30	67
geom40	31.00	31	0	0	0	40	76
geom40a	46.00	46	0	0	0	40	69
geom40b	14.00	14	0	0	0	40	96
geom50	35.00	35	0	0	0	50	96
geom50a	61.00	61	0	0	0	50	106
geom50b	17.00	18	1	0	0	50	121
geom60	36.00	36	0	0	0	60	124
geom60a	65.00	65	0	0	0	60	120
geom60b	22.00	22	0	0	0	60	129
geom70	44.00	44	0	0	0	70	131
geom70a	71.00	71	0	0	0	70	130
geom70b	22.00	23	1	0	1	70	160
geom80	63.00	63	0	0	0	80	130
geom80a	68.00	68	0	0	0	80	168
geom80b	25.00	26	1	0	1	80	211
geom90	51.00	52	1	0	1	90	171
geom90a	65.00	66	1	0	5	90	243
geom90b	28.00	29	1	0	2	90	213
geom100	60.00	60	0	0	1	100	180
geom100a	81.00	81	0	0	2	100	241
geom100b	30.00	31	1	0	25	100	276
geom110	62.00	63	1	0	10	110	212
geom110a	91.00	92	1	0	141	110	260
geom110b	37.00	37	0	0	1	110	214
geom120	63.50	64	0	2	167	120	268
geom120a	93.00	94	1	2	303	120	329
geom120b	34.00	35	1	1	462	120	302

Table 2. Results for Random and Other Miscellaneous Graphs

Instance	LP	Heur	Gap	cpu-lp	cpu-ip	cons	vars
R50-1ga	12.00	12	0	0	0	50	91
R50-1gba	45.00	45	0	0	0	50	82
R50-5ga	28.12	29	0	0	0	50	482
R50-5gba	99.68	100	0	0	0	50	441
R50-9ga	64.00	64	0	0	0	50	253
R50-9gba	228.00	228	0	0	0	50	177
R75-1ga	14.00	15	1	1	1	70	262
R75-1gba	53.00	54	1	0	4	70	224
R75-5ga	37.17	38	0	2	2	75	1290
R75-5gba	130.84	131	0	2	4	75	1262
R75-9ga	93.50	94	0	0	0	75	354
R75-9gba	328.00	328	0	0	0	75	372
R100-1ga	15.00	17*	2	10	1000	100	612
R100-1gba	56.00	57	1	4	127	100	492
R100-5ga	41.96	43	1	7	38	100	2292
R100-5gba	152.57	153	0	6	228	100	2171
R100-9ga	117.29	118	0	0	0	100	786
R100-9gba	421.50	422	0	0	0	100	640
myciel3	10.50	11	0	0	0	11	27
myciel3b	31.50	32	0	0	0	11	24
myciel4	11.71	12	0	0	0	23	83
myciel4b	38.80	39	0	0	0	23	70
myciel5	13.32	14	0	0	0	47	243
myciel5b	44.83	45	0	0	0	47	189
myciel6	15.47	16	0	1	3	95	578
myciel6b	57.14	58	0	2	1	95	595
myciel7	16.37	17	0	30	3	191	1379
myciel7b	60.74	61	0	18	23	191	1096
queen8-8	28.00	29	1	0	1	64	266
queen8-8b	113.00	113	0	0	0	64	148
queen9-9	35.00	36	1	0	2	81	215
queen9-9b	135.00	135	0	0	2	81	242
queen10-10	38.00	40	2	1	123	100	291
queen10-10b	136.00	136	0	0	42	100	282
queen11-11	41.00	44*	3	1	1000	121	349
queen11-11b	140.00	142*	2	3	1000	121	443
queen12-12	42.00	47*	5	38	1001	144	701
queen12-12b	163.0	165.0*	2	1	1000	144	376
DSJC125.1	19.00	21	2	2	57	125	321
DSJC125.1b	67.00	68	1	2	63	125	368
DSJC125.5	52.87	55*	2	20	1001	125	3591
DSJC125.5b	161.5	164.0*	2	19	1000	125	3733
DSJC125.9	139.00	140	1	1	1	125	1388
DSJC125.9b	496.25	497	0	1	0	125	1270

References

[1] Aardal, K.I., S.P.M. Van Hoesel, A.M.C.A. Koster, C. Manino, and A. Sassano. (2001). Models and Solution Techniques for Frequency Assignment Problems 4OR 1:4, 261–317.

[2] Balas, E. and H. Samuelsson. (1977). A node covering algorithm, *Naval Research Logistics Quarterly 24:2*, 213–233.

[3] Balas, E. and J. Xue. (1991). Minimum weighted coloring of triangulated graphs, with application to maximum weight vertex packing and clique finding in arbitrary graphs, *SIAM Journal on Computing 20:2*, 209–221.

[4] Balas, E. and C. S. Yu. (1986). Finding a maximum clique in an arbitrary graph, *SIAM Journal on Computing 15:4*, 1054–1068.

[5] Barnhart, C., E. L. Johnson, G. L. Nemhauser, M. W. P. Savelsbergh, and P. H. Vance. (1998). Branch-and-Price: Column Generation for Huge Integer Programs, *Operations Research 46:3*, 316–329.

[6] Carraghan, C. and P. M. Pardalos. (1990). An exact algorithm for the maximum clique problem, *Operations Research Letters 9*, 375–382.

[7] Coffman Jr., E.G., M.R. Garey, D.S. Johnson, and A.S. Lapaugh. (1985). Scheduling File Transfers *SIAM Journal on Computing 14:4*, 743-780.

[8] Jerrum, M. (1992). Large cliques elude the metropolis process, *Random Structures and Algorithms 3:4*, 347–360.

[9] Johnson, D.S. A. Mehrotra, and M.A. Trick. (2002). Computational Challenge on Graph Coloring and its Generalizations International Symposium on Mathematical Programming, Copenhagen, Denmark.

[10] Johnson, E.L. (1989). Modeling and strong linear programs for mixed integer programming, *Algorithms and Model Formulations in Mathematical Programming, NATO ASI 51*, S.W. Wallace (ed.), Springler-Verlag Berlin, Heidelberg, 1-43.

[11] Khoury, B.N. and P. M. Pardalos. (1996). An algorithm for finding the maximum clique on an arbitrary graph, *Second DIMACS Challenge: Cliques, Coloring, and Satisfiability, DIMACS Series on Discrete Mathematics and Theoritical Computer Science*, D. S. Johnson and M. A. Trick (eds.), American Mathematical Society, Providence.

[12] Mehrotra, A. and M. A. Trick. (1996). A column generation approach for exact graph coloring, *INFORMS Journal on Computing*, **8:4**, 133-151.

[13] Mehrotra, A. and M. A. Trick. (1998). Cliques and Clustering: A Combinatorial Approach, *Operations Research Letters*, **22:1**, 1-12.

[14] Narayanan, L. (2002). Channel Assignment and Graph Multi-coloring, in *Handbook of Wireless Networks and Mobile Computing*, Wiley.

[15] Nemhauser, G.L. and L. E. Trotter. (1975). Vertex packings: Structural properties and algorithms, *Mathematical Programming 8*, 232–248.

[16] Pittel, B. (1982). On the probable behaviour of some algorithms for finding the stability number of a graph, *Mathematical Proceedings of the Cambridge Philosophical Society 92*, 511–526.

[17] Prestwich, S. (2006). Generalized Graph Coloring by a Hybrid of Local Search and Constraint Programming, *Discrete Applied Mathematics*, to appear.

[18] Ryan, D.M. and B.A. Foster. (1981). An integer programming approach to scheduling, in *Computer Scheduling of Public Transport Urban Passenger Vehicle and Crew Scheduling*, North-Holland, Amsterdam, 269-280.

[19] Vanderbeck, F. (2005). Branching in Branch-and-Price: A Generic Scheme, manuscript, Applied Mathematics, University Bordeaux 1, F-33405 Talence Cedex, France.