Efficient Coloring of a Large Spectrum of Graphs

Darko Kirovski and Miodrag Potkonjak
Computer Science Department, University of California, Los Angeles, CA 90095-1596

ABSTRACT

We have developed a new algorithm and software for graph coloring by systematically combining several algorithm and software development ideas that had crucial impact on the algorithm's performance. The algorithm explores the divide-and-conquer paradigm, global search for constrained independent sets using a computationally inexpensive objective function, assignment of most-constrained vertices to least-constraining colors, reuse and locality exploration of intermediate solutions, search time management, post-processing lottery-scheduling iterative improvement, and statistical parameter determination and validation. The algorithm was tested on a set of real-life examples. We found that hard-to-color real-life examples are common especially in domains where problem modeling results in denser graphs. Systematic experimentations demonstrated that for numerous instances the algorithm outperformed all other implementations reported in literature in solution quality and run-time.

1 Introduction

The ever-increasing amount of resources encountered in contemporary designs with exponentially ascending complexities impacts optimization intensive development, compilation, and fabrication strategies. Since many problems related to resource sharing can be modeled as graph coloring, its importance is self evident. Possible applications include register assignment, cache line coloring, logic minimization, circuit testing, operations scheduling, etc [Cou97].

Graph coloring is an NP-complete problem. Previous work shows that so far algorithm development was focused on establishing a single heuristic search methodology expected to provide exceptional results on all instances. Such algorithms usually perform well only on small subset of graph model classes or real-life examples. For example, Johnson et al. [Joh91] simultaneously employed three distinct simulated annealing graph coloring techniques, and one multi-variant successive augmentation XRLF algorithm on a set of various graph models. All implementations resulted in relatively low-quality solutions on some instances.

Recently, Coudert proposed a set of exact algorithms for graph coloring based on the knowledge of the maximum clique in a graph [Cou97]. Statistically, such reduction of a hard problem onto another hard problem is not a good problem solving strategy especially when the reduction result is finding a maximum clique in a graph, a problem that has been suggested, due to its difficulty, to hide information in some cryptographic protocols [Jue96]. Coudert applied his algorithm on a large set of real-life examples. However, he did not provide experimental results for most graph classes which correspond to difficult optimization problems.

Complex problems, such as graph coloring, require extensive exploration of the problem solution space. Deployment of a single search or constraint strategy usually leads to a limited adaptability to particular problem topologies. Moreover, finding strategies that meet search quality requirements for a large number of problem topologies is an extremely hard task. We have developed a new algorithm for graph coloring which outperforms all algorithms reported in literature with respect to both solution quality and search run-time. In this paper, we describe the key algorithm and software development ideas that had crucial impact on the algorithm's performance. The key algorithm development techniques are: divide-and-conquer paradigm (coloring one color class, e.g. one independent set of vertices at a time), global probabilistic search for the best intermediate solution (current color class) and its computationally inexpensive objective function computation, least-constraining and most-constrained heuristics, reuse and locality exploration of intermediate solutions, search timing management, post-processing iterative improvement using lottery-scheduling probabilistic search, and once the software is written, optimization of execution bottlenecks and statistical determination and validation of all parameters in the program.

The algorithm implementation strategy can be generalized for a subset of NP-hard problems such as graph partitioning, and feedback arc set. The algorithm is well tested on a number of real-life examples and graph models which correspond to possible real-life optimization problems. We found that hard-to-color real-life examples exist especially in domains where problem modeling results in denser graphs.

DAC 98, June 15-19, 1998, San Francisco, CA USA
ISBN 1-58113-049-x/98/06…$5.00

2 Related Work

Algorithms for graph coloring can be classified into three main classes: exact [Bre79, Pee83, Cou97], constructive with [Hal93] and without [Bre79, Lei79] guaranteed coloring quality, and iterative improvement coloring algorithms [Bol85, Cha87, Her87, Mor86, Joh91, Mor94, Fle96]. The best results up to date were reported for algorithms that utilize knowledge about the solution space topology in order to establish exceptional colorings. Such algorithms were implemented Johnson et al. [Joh91], Morgenstern [Mor94], etc. An efficient hybrid genetic tabu search algorithm has been developed by Fleurent and Ferland [Fle96]. The standard test example, random $V = 1000$ vertex graph with $p = 0.5$ probability of edge existence, averaged 85.5 for Johnson et al. They required 136 hours on a Sequent Balance 21000 single processor multicomputer. The recent attacks by both Morgenstern, and Fleurent and Ferland resulted in 84-colorings of the $G(1000, 0.5)$ graph, and took between 9 and 36 hours on five Sparc-2s upgraded with 80MHz CPUs for Morgenstern's mix of distributed XRLF and simulated annealing based rejectionless Metropolis move selection, and 41.35 hours on a Sparc-10 workstation for the tabu search genetic algorithm implementation.

3 The New Approach

The problem of graph coloring can be defined using the standard Garey-Johnson format [Gar79]:

PROBLEM: GRAPH K-COLORABILITY

INSTANCE: *Graph $G(V, E)$, positive integer $K \leq |V|$.*

QUESTION: *Is G K-colorable. i.e., does there exist a function $f : V \rightarrow 1, 2, 3, .., K$ such that $f(u) \neq f(v)$ whenever $u, v \in E$?*

We introduce two novel distinct graph coloring algorithms and propose their hybrid as a champion algorithmic solution to the graph coloring problem. The first algorithm is a successive augmentation of a global probabilistic **l**east-constraining **m**ost-constrained e**X**tended RLF (lmXRLF). The second is **l**ottery-**s**cheduling-driven **I**terative **I**mprovement algorithm (lsII). Their hybrid (lmXRLF/lsII) uses initially the lmXRLF algorithm to reduce the solution search space to a statistically determined reduced graph size checkpoint. Then, the lsII algorithm iteratively improves the solution until the user stops the search for improvement.

The performance evaluation and validation of a graph coloring algorithm is strongly dependent on the benchmark graph topologies. We used several diverse graph models: random, geometrical, k-cooked, and uniquely k-cooked graph model. The details about the generation algorithms for each of the used graph models are given in [Kir98].

3.1 lmXRLF

Problem solution space reduction is essential in fast solving of problems with large domains such as graph coloring. The lmXRLF algorithm performs an efficient solution space reduction by assigning the most-constrained vertices for coloring to the class of least-constraining colors, i.e. colors assigned early in the successive coloring process. The remaining vertices in the graph after each color assignment are expected to establish a graph coloring subproblem quantitatively and qualitatively easier to solve than the one in the previous step. Successive augmentation of the described technique, that colors the most-constrained vertices with the least-constraining color, evolves into a complete lmXRLF algorithm. The objective function that quantifies the vertex constraints has crucial impact on the algorithm's performance. We have determined experimentally the least-constraining most-constrained objective of this search. The pseudo-code of the algorithm is presented in Figure 1. In the rest of this subsection we describe the lmXRLF algorithm in detail.

The outermost loop of the algorithm drives the search for the most objective-efficient independent set. Once the underlying search returns the subset, all vertices in the independent set are colored with the next available color and deleted from the graph. The described graph reduction is unchangeable, i.e., deleted vertices are not revisited. The objective function that decides the independent set selection was experimentally determined and validated on a set of examples [Kir98]. The objective function found to perform the best on a set of benchmarks is:

$$ObjF(S) = \sum_{V \in S} (NoN(V)^2 \cdot \sum_{U \in Neigh(V)} NoN(U)^3)$$

where S is the evaluated independent set, $Neigh(V)$ returns the set of vertices adjacent to V, and $NoN(V)$ returns the number of vertices adjacent to V. Thus, vertices found to have large number of incident edges as well as neighborhood with large number of neighbors are treated as constrained. Finding a set of vertices which has the largest accumulated sum of constraints corresponds to the INDEPENDENT SET problem [Gar79].

Important property of the objective function is its local computation and usage of simple search operations such as *remove node* and *add node* from/to the current independent set. Local cost computation means that the objective function is not computed over the entire graph, but only over the nodes in the independent set. The random numbers and their squares and cubes used in the computation are precomputed to avoid complex arithmetic operations. This approach enables fast independent set evaluation and boosts the number of independent sets considered in the search.

The search for the most objective-efficient independent set is performed in the following way. The algorithm first creates a random initial independent set S. Then it iteratively randomly deletes vertices from S until there exists at least one node in the graph not adjacent to the remaining vertices in the set. The uncovered vertices are then randomly selected and added to S until it becomes an independent subset. We experimented greedy strategies for vertex

exclusion/inclusion but they did not yield any performance improvement over the fully probabilistic search. Besides introducing larger computation delays between generating two independent sets, controlling the convergence of the greedy probabilistic search strategies is time-consuming and highly sensitive to graph topologies [Joh91]. On the other hand, fully probabilistic search showed unexpectedly good results for all experimented instances.

```
Color = 0
Repeat
  If Color = 0 then GlobalIterations = F · GLOBAL
  Else GlobalIterations = GLOBAL · (1 − ColoredVerts^2 / Verts^2)
  Generate a random independent set S and add it to the list of
  solutions ListOfBestSolutions with largest objective costs
  Repeat GlobalIterations times
    Delete randomly vertices from S until at least one vertex does
    not have a colored neighbor (not counting the deleted ones)
    Add randomly vertices which do not have colored neighbors to S
    If ObjF(S) > min(ObjF(ListOfBestSolutions_i)), i = 1..F
      Add S to the list
      Discard solution S_x with min(ObjF(S_x) from the list
  End Reapeat
  For each independent set S in the ListOfBestSolutions
    Repeat
      S+ = S
      Repeat LOCAL times
        S* = S
        Delete randomly vertices from S* until at least one vertex
        does not have a colored neighbor (not counting the deleted ones)
        Add randomly vertices which do not have colored neighbors to S*
        If ObjF(S*) > ObjF(S+) then S+ = S*
      S = S+
    until there is at least one improvement
    End Reapeat
  End For
  For each V_i ∈ max(ObjF(ListOfBestSolutions_i), i = 1..F)
    COLOR V_i with Color
    Discard V_i from further color consideration
  End For
  Delete all solutions S ∈ ListOfBestSolutions with at least one
  common V_i with max(ObjF(ListOfBestSolutions_i), i = 1..F)
  Color = Color + 1
until there are uncolored vertices
End Repeat
```

Figure 1: Pseudo-code of the lmXRLF algorithm.

During the problem reduction search, at each step a number of independent sets is generated. The search is terminated when $GlobalIterations$ successive randomly generated independent sets do not yield any solution improvement. Due to the randomized search, the locality of the most objective efficient color class is searched extensively in order to find better solution. While searching locally, $LOCAL$ number of independent sets are generated starting from the solution returned by the global search.

While searching for the most objective efficient independent set, a set of color class candidates is generated. When an extensive search is performed, candidates may have disjoint color classes. In order not to discard the solutions which may be objective-competitive and most probably not reachable in the next search iteration, a list of F most objective-efficient solutions is maintained throughout the search. When the search is stopped and the best independent set is colored and deleted from the graph, all other independent sets in the list are checked for mutual vertices and if conjunctive, deleted from the list. The remaining independent sets are a good starting point for the next search iteration.

Since the first iteration in the color class search is handicapped due to lack of previous results, the number of search iterations is adjusted to the number of solutions available from previous searches and the cardinality of the remaining graph. For the first color class $F \cdot GLOBAL$ candidates are generated. For every succeeding color class, the number of generated candidates during the global search equals $GLOBAL \cdot (1 - \frac{ColoredVerts^2}{Verts^2})$. Therefore, the program search time is managed in such a way that difficult to find subsets in the beginning of the search are given longer timeouts. A visual example how the search is performed is presented in [Kir98].

3.2 lsII

There exist instances (e.g. small or extremely dense graphs), for which iterative solution improvement algorithms, that use probabilistic search to traverse the solution space, perform exceptionally well. While searching, the best colorings are memorized and successive trials are conducted in order to improve upon the best memorized solution. At any time during the search, the vertex selection and coloring is non-deterministically greedy, i.e. some probability of traversing towards worse solutions always exists. The main disadvantage of employing such algorithms is the fact that they do not have fixed run-times. Moreover, the run-times for obtaining a solution may differ substantially from one graph to another with the same cardinalities and topology. Finally, search parameters, such as annealing rate, are difficult to establish and greatly vary depending on the graph topology [Joh91].

We developed a novel iterative improvement algorithm for graph coloring. The lsII algorithm tries to color the graph with K colors. The graph is initially, usually incorrectly, colored randomly using K colors. The incorrectly colored vertices are considered for recoloring using the steepest descent rule and the lottery-scheduling paradigm for vertex and color selection probability distribution [Wal94]. The recoloring step is iterated until feasible coloring is found. The algorithm is described using pseudo-code in Figure 2.

Each recoloring step consists of two phases. In the first phase, for each incorrectly colored vertex an objective is assigned describing the vertex' constraint. An incorrectly colored vertex is selected using a lottery scheduling driven probabilistic selection such that vertices with higher constraints are given proportional selection priority. As an objective function to quantify the vertex constraint, we used the number of adjacent vertices colored with the same color as the incorrectly colored vertex. In the second phase, the colors are assigned objectives which correspond to their constraints. The constraint of a color is computed according to the following formula:

$$ObjF(V, C) = \frac{1}{1 + VertsAdjTo(V, C)^{\alpha}}$$

where α is fixed parameter and $VertsAdjTo^*(V, C)$ returns the number of vertices adjacent to V and colored with C. According to the individual constraint of each color lottery scheduling driven probabilistic selection is used to select the one in which the selected incorrectly colored vertex is to be colored. Priority is given to colors rarely used in the incorrectly colored vertex' neighborhood. The algorithm iteratively performs the recoloring phases until a viable coloring is found.

Randomly color all V vertices with K colors
Repeat
  Generate random number
    $R \in [0, \sum_{V \in ICVerts(G)} VertsAdjTo(V, V.Color)$
  Select vertex V_w such that
    $R \geq \sum_{j=V_0..V_{[w-1]} \in ICVerts(G)} VertsAdjTo^*(V_j, V_j.Color)$
  and w is minimum
  Generate random number
    $Q \in [0, \sum_{i=1..COLORS} \frac{1}{1+VertsAdjTo(V_w,i)^\alpha}$
  Select color C_x such that
    $Q \geq \sum_{j=C_0..C_{[x-1]}} VertsAdjTo(V_w, C_j)$ and x is minimum
  Color V_w with C_x
  If $\sum_{V \in ICVerts(G)} VertsAdjTo(V, V.Color)$
  of the current coloring G is minimum
    Memorize G as the best coloring G^+
  If there is no improvement of
    $\sum_{V \in ICVerts(G)} VertsAdjTo(V, V.Color)$
  after $SUCC$ successive colorings
    Color all vertices with colors of G^+
until graph colored properly

$ICVerts(G)$ returns the set of incorrectly colored vertices in the graph G.
$VertsAdjTo(V, V.Color)$ returns the number of vertices adjacent to V colored with color $V.Color$.

Figure 2: Pseudo-code of the lsII algorithm.

The probability distribution using lottery scheduling is accomplished in the following way. A binary tree is built such that each candidate is assigned to one leaf. The nodes of the tree are assigned the sum of objectives of all leaves contained in the subtree to which the current node is the root. Then, a random number R is selected uniformly in the range of the objective $ObjF(root)$ assigned to the root of the entire tree. According to this number the tree is depth-search traversed to the left child if $R < ObjF(currentNode)$ or right otherwise, until a particular leaf is reached. The reached leaf is the lottery winner and the selected candidate. Implementation details for efficient lottery scheduling are presented in [Wal94].

Parameter α has strong impact on the convergence speed of the algorithm. Large values leave no room for non-greedy recolorings. This results in fast but inflexible convergence and usually oscillations. Smaller values result in longer convergence, but explore the search space more extensively. In our experiments, we dynamically converged α from starting $\alpha = 10$ to asymptotic $\alpha = 4$, and $SUCC = 1000$. An example how lottery scheduling is used in order to achieve probabilistic color perturbation is presented in [Kir98].

The developed lsII iterative improvement algorithm performs exceptionally well on graphs with relatively small number of edges (see Table 1). Graphs with large number of edges induce non-tolerable run-times which make the algorithm impractical for such graphs. General characteristic of all improvement algorithms is that their run-time is highly unpredictable. Standard deviations of typical run-times reported in literature reach the order of magnitude of the average search run-time [Mor94]. Therefore, we employ this algorithm only to small graph instances where the effect of extremely variant expected run-times can be neglected.

3.3 The Hybrid lmXRLF/lsII Algorithm

In order to explore the benefits of both the constructive and iterative improvement algorithm, we coordinated their application to the graph coloring problem. The flow of the algorithm is depicted in Figure 3. The lmXRLF algorithm is used to reduce the search space by peeling sets of color classes until a graph of edge cardinality equal to or less than CAR remains. Then, the lsII algorithm is invoked. The starting chromatic number K for the lsII algorithm is obtained as a result of the application of the lmXRLF algorithm to the remaining subgraph. Every time the lsII returns successful coloring it, is reinvoked with decremented trial chromatic number K. The coloring process is either timed-out or terminated by the user.

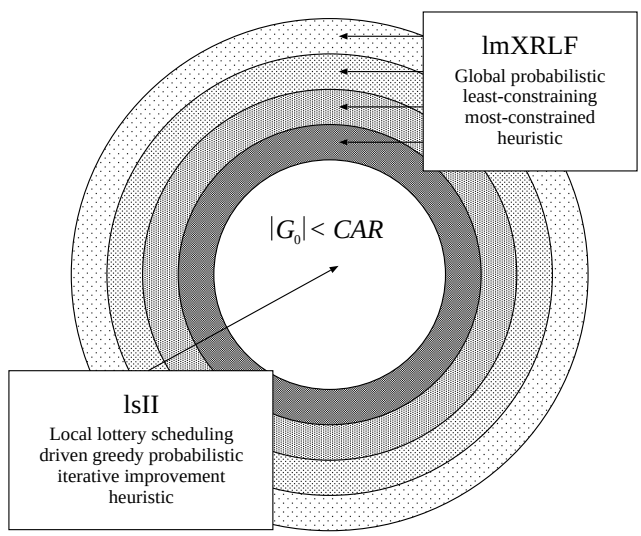

Figure 3: The global algorithm flow: coordinated lmXRLF/lsII deployment for graph coloring.

4 Experimental Results

In this section the conducted experiments are described and results are presented. Obtained colorings are presented for sets of random, k-cooked, uniquely colorable k-cooked, geometric [Kir98], real-life, and DIMACS benchmark graphs.

The independent set selection function was determined and validated by performing a statistic evaluation of performance measurements for a set of objective function candi-

dates [Kir98]. Function $ObjF$ (see subsection 3.1) appeared to have statistically the best performance from the set of function candidates. Therefore, in the rest of the conducted experiments objective-efficient independent sets were evaluated using this function.

Table 1 shows the experimental results for coloring a set of random $R(V,p), p \in \{0.5, 0.1, 0.9\}$ graphs. First two columns describe the $G = GLOBAL$ and $L = LOCAL$ properties of the lmXRLF algorithm or signify that the lsII algorithm was used. The size of the window used to store the best independent sets was $F = 10$ for lmXRLF. Next eight columns present obtained average colorings and run-times in seconds for a set of 10 graph instances for each vertex count. The timings are shown with respect to the $GLOBAL$ and $LOCAL$ parameters. The simulations of the instances with vertex count equal to 125, 250, and 500 were executed on a 40MHz Sparc4, while the 1000-node graphs were colored on a 200MHz UltraII processor. Rows detoned with $\chi(G)$ specify probabilistically determined chromatic number of each graph.

Trials	$R(125,0.5)$		$R(250,0.5)$		$R(500,0.5)$		$R(1000,0.5)$	
$\chi(G)$	16		27		46		80	
G - L	Cols	Time	Cols	Time	Cols	Time	Cols	Time
1 - 1	21	0.08	35.6	0.33	61.8	2.16	113	3.28
10 - 10	18.9	0.80	31.1	3.27	53.9	18.3	-	-
100 - 100	18.3	5.18	30.7	21.9	51.8	117	-	-
10^3 - 100	18.3	6.41	30.3	25.3	51.1	133	90.6	150
10^4 - 100	18.2	9.67	30.2	35.3	50.9	174	-	-
10^5 - 100	18.2	46.2	29.9	172	50	625	85.9	2850
10^6 - 100	-	-	30	420	49.7	4612	85.1	18083
lsII	18	0.23	30	112	-	-	-	-
	17	117	29	-	-	-	-	-
Trials	$R(125,0.1)$		$R(250,0.1)$		$R(500,0.1)$		$R(1000,0.1)$	
$\chi(G)$	5		7		11		19	
1 - 1	7.7	0.01	11.5	0.03	18.2	0.16	29.3	0.25
10 - 10	6	0.17	9.1	0.79	14.9	5.53	-	-
100 - 300	6.1	2.28	9.2	9.47	14.4	64.2	-	-
10^3 - 300	6	2.54	9	10.1	14	65.8	23	30.4
10^4 - 300	6	3.31	9	12	14.1	72.3	-	-
10^5 - 300	6	13.9	9	38.7	13.9	172	22	1319
10^6 - 300	-	-	9	289	13.3	1045	21.8	9079
lsII	6	0.01	9	0.86	13	471	22	4516
	5	3.23	8	14.6	12	-	21	-
Trials	$R(125,0.9)$		$R(250,0.9)$		$R(500,0.9)$		$R(1000,0.9)$	
$\chi(G)$	40		70		122		217	
1 - 1	48.1	0.14	86.4	0.7	154	6.7	279	14.4
10 - 10	46.2	0.9	78	3.7	137	32	-	-
100 - 300	46	17	77	68	133	424	-	-
10^3 - 300	45.8	19	77.5	77	133	511	234	904
10^4 - 300	46	24.4	77.7	97	133	791	-	-
10^5 - 300	45.8	96.5	77.3	413	132	4481	231	10935
lsII	43	2.25	73	1117	-	-	-	-
	42	118	72	-	-	-	-	-

Table 1: Average random $p \in \{0.5, 0.9, 0.1\}$ graph colorings using lmXRLF and lsII.

During the simulations, for the 125-node graphs, we succeeded to obtain 17-colorings on two graphs for two different number of trials (10^5 and $3 \cdot 10^5$). For the set of 250-vertex graphs, we succeeded to obtain 29 coloring only for one graph, while for all others, we obtained 30-colorings. The 500-node graphs were all colored with 50 colors after less than 10 minutes, while more extensive searching resulted in three 49-colorings. We obtained 85-colorings after 30-minutes and reached 85.4 average after less than 1.9 hours on a sample of ten 1000-node graphs. We obtained one 84-coloring in 1.9 hours and two in 5+ hours. The lsII algorithm was applied only to the set of 125- and 250-vertex graphs. Although significantly better results were obtained on smaller graphs than with lmXRLF, competitive coloring of 500-vertex and larger graphs took more than 5+ hours. Moreover, finding 17-colorings on 125-graphs took from 5 to 300 seconds. In one out of twenty cases, the algorithm could not find a 17-coloring in less than 1 hour.

Using the hybrid lmXRLF/lsII algorithm, we were always able to obtain 84-colorings of our set of $R(1000, 0.5)$ graphs. The 84-colorings were obtained even for 10^5 trials of lmXRLF and $CAR = 8000$ for lsII. The obtaining of 84-colorings averaged 50-minutes.

We succeeded to obtain one 12- and two 21-colorings for the $R(500, 0.1)$ and $R(1000, 0.1)$ graphs. Johnson et al. reported for their XRLF implementation that it took 16.5 hours to get 13-coloring for $R(500, 0.1)$ and 35 hours for a 22-coloring of $R(1000, 0.1)$. The lmXRLF did not perform well for $p = 0.9$ type of graphs. lsII demonstrated more powerfull search capabilities on such graph samples.

p	$G = 1000$; $L = 100; F = 6$				$G = 100000$; $L = 100; F = 8$			
k	32	64	Single 64	Single 96	32	64	Single 64	Single 96
0.1	32	64	-	-	32	64	-	-
0.2	32	64	-	-	32	64	-	-
0.3	33.7	64	71.2	-	32.5	64	71.5	-
0.4	38.2	65	74.7	102	32	64	73	102
0.5	32.2	65.7	77.7	105.7	33	65	75.5	105.5
0.6	32	66.2	77.5	109.7	33.5	64.5	71	109.5
0.7	32	66	64.5	113	32	66	64.5	111
0.8	32	64.5	64	102	32	64	64	103
0.9	32	64	64	96.5	32	64	64	96.5

Table 2: Average k-cooked and uniquely colorable k-cooked graph colorings using lmXRLF.

The algorithm performance for coloring k-cooked and single solution k-cooked 500-node graphs is presented in Table 2. Table 2 presents obtained colorings for a set of 500-node with fixed k and variable p. Single solution graphs are created for two values of $k = 64, 96$, while regular cooked graphs are created for $k = 32, 64$. The table indicates the correlation between k, p, and the coloring difficulty when the solution is unique.

The lmXRLF algorithm was applied to the challenge graphs at the DIMACS site. Tables 3 and 4 contain a report on the obtained colorings using three different sets of search parameters. In both tables, the first column represents the file name. Next three columns represent graph's vertex and edge cardinalities, and the optimal coloring if known. Next three columns represent three different search trials with different parameter sets. Note that none of the coloring runs did not take more than 1.5 hours on a Sparc4. We report 84-coloring in 50 minutes of the DSJC1000.5 graph when lmXRLF/lsII algorithm was applied. Similarly to Coudert's

experimental results [Cou97], we were able to solve ALL real-life examples at the DIMACS site optimally in less than a second (all parameters equal 1, except in few cases where $G = 100$). However, difficult-to-color real-life optimization problems exist, in particular, when the conceptual problem modeling results in higher vertex degrees. We present few examples extracted from the cache line coloring optimization challenge [Kir98].

$FILENAME$	V	E	k	A	B	C
DSJC1000.1	1000	99258	?	29	22	22
DSJC1000.5	1000	499652	?	116	89	85
DSJC125.1	125	1472	?	8	6	6
DSJC125.5	125	7782	?	22	19	18
DSJC125.9	125	13922	?	50	45	45
DSJC250.1	250	6436	?	11	9	9
DSJC250.5	250	31336	?	39	30	30
DSJC250.9	250	55794	?	85	77	77
DSJC500.1	500	24916	?	19	14	14
DSJC500.5	500	125248	?	64	51	50
DSJC500.9	500	224874	?	156	134	134
DSJR500.1	500	7110	?	15	13	13
DSJR500.1c	500	242550	?	97	96	96
DSJR500.5	500	117724	?	144	128	128
anna	138	986	11	11	11	11
david	87	812	11	11	11	11
flat1000_50_0	1000	245000	50	112	87	50
flat1000_60_0	1000	245830	60	112	89	61
flat1000_76_0	1000	246708	76	112	89	85
flat300_20_0	300	21375	20	39	20	20
flat300_26_0	300	21633	26	41	31	28
flat300_28_0	300	21695	28	41	33	32
fpsol2.i.1	496	11654	65	65	65	65
fpsol2.i.2	451	8691	30	30	30	30
fpsol2.i.3	425	8688	30	30	30	30
games120	120	1276	9	9	9	9
homer	561	3258	13	14	13	13
huck	74	602	11	11	11	11
inithx.i.1	864	18707	54	54	54	54
inithx.i.2	645	13979	31	31	31	31
inithx.i.3	621	13969	31	31	31	31
cacheline.1	822	22117	?	64	48	46
cacheline.3	585	9912	?	37	34	31
$A \rightarrow GLOBAL = 1; LOCAL = 1; F = 1$						
$B \rightarrow GLOBAL = 1000; LOCAL = 100; F = 6$						
$C \rightarrow GLOBAL = 100000; LOCAL = 100; F = 8$						

Table 3: lmXRLF-Algorithm performance for the DIMACS set of challenge graphs.

5 Conclusion

We have developed a novel algorithm design strategy which focuses on building an effective software implementation of a mix of various constraint quantification and problem search strategies rather than using a single heuristic. We demonstrated the efficacy of the strategy on the graph coloring problem. Extensive experimentations showed that the algorithm outperforms all other implementations reported in literature for order of magnitude in search run-time while maintaining the solution quality.

6 References

[Bol85] Bollobas, B., Thomason, A. Random Graphs of Small Order. Annals of Disc. Math., vol.28, pp. 47-57, 1985.

[Bre79] Brelaz, D. New methods to color the vertices of a graph. Comm. of the ACM, vol.22, (no.4), pp. 251-6, 1979.

[Cha87] Chams, M., et al. Some Experiments With Simulated Annealing for Coloring Graphs, European Journal of Operations Research, vol.32, pp.260-66, 1987.

[Cou97] Coudert, O. Exact coloring of real-life graphs is easy. 34th Design Automation Conference, pp. 121-126, 1997.

[Fle96] Fleurent, C., Ferland, J.A. Genetic and hybrid algorithms for graph coloring. Annals of Operations Research, vol.63, pp.437-461, 1996.

[Gar79] Garey, M.R., Johnson, D.S. Computers and intractability: a guide to the theory of NP-completeness. W. H. Freeman, San Francisco, CA, 1979.

[Hal93] Halldorsson, M.M. A still better performance guarantee for approximate graph coloring. Information Processing Letters, vol.45, (no.1), pp.19-23, 1993.

[Her87] Hertz, A., de Werra, D. Using Tabu Search Techniques for Graph Coloring, Computing, vol.39, pp.345-351, 1987.

[Joh91] Johnson, D.S., et al. Optimization by simulated annealing: an experimental evaluation. II. Graph coloring and number partitioning. Operations Research, vol.39, (no.3), pp.378-406, 1991.

[Jue96] Juels, A., Peinado M. Hidden cliques as cryptographic keys. Technical report, UC Berkeley, 1996.

[Kir98] Kirovski, D., Potkonjak, M. Exact Coloring of Many Real-Life Graphs is Difficult, but Heuristic Coloring is Almost Always Effective. Technical Report, CSD, UCLA, 1997.

[Lei79] Leighton, F.T. A Graph Coloring Algorithm for Large Scheduling Algorithms. Journal of Res. Natl. Bur. Standards, vol.84, pp.489-506, 1979.

[Mor86] Morgenstern, C., Shapiro, H. Chromatic Number Approximation Using Simulated Annealing. Unpublished, 1986.

[Mor94] Morgenstern, C. Distributed Coloration Neighborhood Search. DIMACS Series in Disc. Math., vol.0, 1994.

[Pee83] Peemoeller, J. A correction to Brelaz's modification of Brown's coloring algorithm. Communications of the ACM, vol.26, (no.8), pp.595-7, 1983.

[Wal94] Waldspurger, C.A.; Weihl, W.E. Lottery scheduling: flexible proportional-share resource management. The First USENIX Symposium on Operating Systems Design and Implementation, pp.1-11, 1994.

$FILENAME$	V	E	k	A	B	C
latin_square_10	900	307350	?	129	109	100
le450_15a	450	8168	15	21	17	17
le450_15b	450	8169	15	21	17	17
le450_15c	450	16680	15	29	22	21
le450_15d	450	16750	15	29	22	21
le450_25a	450	8260	25	26	25	25
le450_25b	450	8263	25	27	25	25
le450_25c	450	17343	25	33	28	28
le450_5a	450	5714	5	12	7	5
le450_5b	450	5734	5	13	7	5
le450_5c	450	9803	5	16	5	5
le450_5d	450	9757	5	16	5	5
miles1000	128	6432	42	45	43	42
miles1500	128	10396	73	75	73	73
miles250	128	774	8	9	8	8
miles500	128	2340	20	20	20	21
miles750	128	4226	31	33	32	32
mulsol.i.1	197	3925	49	49	49	49
mulsol.i.2	188	3885	31	31	31	31
mulsol.i.3	184	3916	31	31	31	31
mulsol.i.4	185	3946	31	31	31	31
mulsol.i.5	186	3973	31	31	31	31
myciel7	191	2360	8	8	8	8
queen15_15	225	10360	?	20	17	17
queen16_16	256	12640	?	23	18	18
school1	385	19095	14	39	16	14
school1_nsh	352	14612	14	35	16	14
zeroin.i.1	211	4100	49	49	49	49
zeroin.i.2	211	3541	30	31	30	30
zeroin.i.3	206	3540	30	30	30	30
cacheline.2	1121	287220	?	74	53	50
cacheline.4	742	15001	?	44	37	35
$A \rightarrow GLOBAL = 1; LOCAL = 1; F = 1$						
$B \rightarrow GLOBAL = 1000; LOCAL = 100; F = 6$						
$C \rightarrow GLOBAL = 100000; LOCAL = 100; F = 8$						

Table 4: lmXRLF-Algorithm performance for the DIMACS set of challenge graphs.