Experiments with Parallel Graph Coloring Heuristics

Gary Lewandowski
Anne Condon

Technical Report #1213

February 1994

Experiments with Parallel Graph Coloring Heuristics

Gary Lewandowski [*]
Anne Condon[†]
Computer Science Department
University of Wisconsin at Madison

January 1994

Abstract

We report on experiments with a new hybrid graph coloring algorithm, which combines a parallel version of Morgenstern's S-Impasse algorithm [20], with exhaustive search. We contribute new test data arising in five different application domains, including register allocation and class scheduling. We test our algorithms both on this test data and on several types of randomly generated graphs. We compare our parallel implementation, which is done on the CM-5, with two simple heuristics, the Saturation algorithm of Brélaz [4] and the Recursive Largest First (RLF) algorithm of Leighton [18]. We also compare our results with previous work reported by Morgenstern [20] and Johnson et al. [13]. Our main results are as follows.

- On the randomly generated graphs, the performance of Hybrid is consistently better than the sequential algorithms, both in terms of speed and number of colorings produced. However, on large random graphs, our algorithms do not come close to the best colorings found by other time-intensive algorithms such as the XRLF algorithm of Johnson et al. [13] and Morgenstern's tuned S-Impasse algorithm.
- Of the five types of test data, three are easily colored even by the simple RLF and Saturation heuristics; one (the class scheduling data) is optimally colored by Hybrid but not by the simple heuristics, and one appears to be very hard. However, it should not be concluded that coloring is "easy" in most applications. In several cases, such as the class and exam scheduling graphs, finding an optimal coloring is not sufficient to solve the problem at hand, but rather colorings satisfying additional restrictions are really needed.
- The Hybrid algorithm parallelizes well. This appears to be for three main reasons. First, the number of iterations needed by the S-impasse decreases as the number of processors increase. Second, in the exhaustive search algorithm the work involved in expanding the search tree is effectively shared among the processors. Third, on some tests, the S-Impasse and Exhaustive-Search procedures progress in a symbiotic fashion, one using a good coloring obtained by the other as a basis for further improvement.

Overall, we are satisfied that our parallel algorithm effectively exploits the processing power of the CM-5, and that further work on a hybrid algorithm can lead to even better results. Since the performance of all the implemented algorithms on random graphs does not correlate well with their performance on application data, we conclude that further effort spent in collecting application data is well justified, and suggest that new generators, which model the structure of application data, be investigated.

[*]lewandow@cs.wisc.edu. Work supported by NSF grants numbers CCR-9257241 and DCR-9208639
[†]condon@cs.wisc.edu. Work supported by NSF grant number CCR-9257241

1 Introduction

A classical problem in graph theory, the graph coloring problem is to color the nodes of an undirected graph with as few colors as possible, such that no adjacent nodes share a color. The earliest reference to the problem is the conjecture of Guthrie in 1852 that all maps (planar graphs) can be colored using no more than four colors. Subsequent work on that problem not only spurred the development of graph theory, but ultimately led to the famous four-color theorem of Appel and Haken [1], based on an extensive computer search.

Graph coloring has many uses beyond map coloring, and one would expect that computers can be used to solve not just the four-color problem, but general graph coloring problems. It is an abstraction of time-tabling problems, in which lists of courses desired by students are given, and the minimal number of class periods such that all students can take their desired courses must be determined (de Werra [7], Leighton [18], Opsut and Roberts [21]). Garey, Johnson and So [8] showed that graph coloring can be used for short circuit testing for printed circuits. Chaitin [5, 6] reduced register allocation to graph coloring and used it in a compiler. Poole and Ortega [22] showed how to use graph coloring to decompose matrices used to solve sparse systems of linear equations; the decomposition gives a method for easy parallelization of the solution.

Given the practical importance of the graph coloring problem, it is unfortunate that, in theory at least, the cards are stacked against the designer of graph coloring algorithms. Not only is it NP-complete to determine if a graph can be colored with a given number of colors [15], but it is also hard even to approximate the chromatic number of a graph. Lund and Yannakakis [19] proved that for some $\epsilon > 0$, approximating the chromatic number within a factor of n^ϵ is NP-hard. The best known approximation algorithm, due to Halldórsson [12], provides an extremely poor performance guarantee of $O(n(\log\log n)^2/(\log n)^3)$ for an n-node graph. (The performance guarantee is the maximum ratio, taken over all inputs, of the number of colors used over the chromatic number).

On the other hand, the results of Grimmet and McDiarmid [10] on coloring algorithms for random graphs offer the algorithm designer some reason for optimism. They consider random graphs such as the $G_{n,p}$ graphs, which have n nodes, where each pair is connected with independent probability p. Their algorithms run in polynomial time and expect to use no more than $2\chi(G)$ colors, where $\chi(G)$ is the chromatic number. However, the hidden constants in the running times make their algorithms prohibitively expensive in practice. Also of course, there is no guarantee that algorithms which work well on random graphs will also work well on data arising from real applications.

There is a fairly extensive body of literature on experimental graph coloring algorithms. These have been tested primarily on random graphs, such as the $G_{n,p}$ graphs mentioned above. One of the simplest coloring algorithms is the Saturation algorithm of Brélaz [4]. It is based on the following principle: the vertex which is adjacent to the greatest number of differently-colored neighbors is colored first, with a new color if necessary. Thus, if ever the color of a vertex is forced, that is, there is at most one possible choice from the current set of colors, that vertex is colored first. Another example of such a "successive augmentation" algorithm is the Recursive Largest First (RLF) algorithm, proposed by Leighton [18] when studying the exam scheduling problem at Princeton. This algorithm colors the vertices one color class at a time, adding vertices one at a time to the current color class so as to reduce as much as possible the number of edges left in the uncolored subgraph.

These two algorithms have very efficient implementations, but as we will see, do not produce very good colorings on standard test data. Johnson et al. [13] pushed the successive augmentation approach much further with the XRLF algorithm, which is essentially a semi-exhaustive version of Leighton's RLF algorithm, based on ideas of Johri and Matula [14]. The XRLF algorithm finds better colorings than the simpler successive augmentation algorithms on random $G_{n,p}$ graphs, but takes significantly more time and is beaten by the simpler Saturation algorithm on other randomly generated classes of graphs.

A quite different approach has been taken with iterative improvement algorithms, which include the simulated annealing algorithm of Johnson et al. [13], the Tabu Search algorithm of Glover [11] and the S-Impasse algorithm of Morgenstern [20]. Briefly, iterative improvement algorithms differ from successive augmentation algorithms in that the colors of individual nodes may change several times over the course of the algorithm. Of the iterative improvement algorithms, Morgenstern reports the best results for the S-Impasse algorithm. However, there are several parameters in this algorithm that need to be tuned for different problem instances. We describe his algorithm in more detail later, as our approach is based on his.

Based on reports of this previous work, we draw the following conclusions. It is hard to design a "robust" graph coloring algorithm, that is, one which works well on a wide range of inputs: none of the current approaches clearly dominates the others. The best algorithms, such as XRLF or S-Impasse, are quite sophisticated, with several parameters that need to be tuned, based on knowledge of the input, or by trial and error. Moreover, in order to get good results, sequential implementations take several hours on a standard workstation, even on relatively small graphs of 1,000 nodes or less, and the time to reduce the size of a coloring by 1 increases greatly as better colorings are found. Finally, there is little experimental work on data from real applications of graph coloring.

One natural approach towards overcoming the limitations of previous algorithms is with a parallel implementation, and this is the approach we take in this project. The potential for good speedup is clear. A MIMD environment also provides one way to achieve robustness, namely to run different algorithms on different processors and to take the best solution found. This hybrid approach has the potential for the whole to be greater than the sum of the parts. For example, the different algorithms can progress in a symbiotic fashion, one using a good coloring obtained by another as a basis for further improvement. Finally, it may be possible to dispense with tuning, since if the parallel implementation is fast enough, the time it spends on converging to good parameter values may be significantly less than that spent by an implementer in finding the right settings.

We describe experiments with a parallel algorithm, Hybrid, which combines a parallel version of Morgenstern's S-Impasse algorithm with an exhaustive search algorithm. Our algorithm is run on the CM-5, a powerful state-of-the-art parallel computer which gives much flexibility in designing hybrid algorithms. We compare the performance of this algorithm with RLF and Saturation, and with previous results reported by Morgenstern and Johnson et al. Our parallel algorithm outperforms the sequential algorithms, both in the quality of the colorings obtained and in the time spent to obtain the colorings, on all but the simplest test graphs, where all algorithms find an optimal coloring very quickly. However, both Morgenstern's tuned sequential implementation of S-Impasse [20] and Johnson et al.'s XRLF algorithm find better colorings than our algorithm on large $G_{n,p}$ graphs. (recall that our tests of Hybrid are untuned). Our conclusion is that our parallel methods are certainly useful in solving the graph coloring problems, but that tuning of the parameters may help further in reducing the running times.

We have collected new test data for our experiments. One test graph is an example of a class scheduling graph, which models the problem of constructing a timetable that allows all students to take their desired courses. The data was obtained from a local high-school, yielding a graph with approximately 400 nodes. Unlike the simpler hueristics, our hybrid algorithm performed extremely well on the class scheduling graph data, finding an optimal coloring very quickly. Another set of test data that we have contributed is based on the problem of register allocation, which arises in compiling programs.

Further data, relating to sparse matrix computations, a problem on latin squares, and an exam schedule, are also contributed.

We describe our parallel algorithm in detail in Section 2. Our test data, including both our new contributions and other, randomly generated data, is described in Section 3. The quality of the colorings obtained for our test data, and the running times of our parallel algorithm are discussed in Section 4. A brief concluding summary and directions for future work are presented in Section 5.

2 Algorithm Description

We first briefly review Morgenstern's S-Impasse algorithm [20]. We then describe a parallel version of the S-Impasse algorithm, a parallel exhaustive search algorithm, and finally our Hybrid algorithm.

The S-Impasse Algorithm. This is an example of an iterative improvement algorithm, proposed by Morgenstern [20]. In the following description, the parameters of the algorithm are italicized.

Initially, a *target* number of color classes is chosen, and a naive coloring of the graph is quickly computed. All vertices from color classes beyond the target are placed in an impasse set. The algorithm then repeatedly does the following. If the current best coloring uses less than the target number of colors, then the target is reduced to one less than the current coloring and a new impasse class is created. (This is different from Morgenstern's original algorithm, which halted upon reaching its target. The user can then choose to rerun the algorithm with a smaller target. We believe our method is more useful to a user.) A random vertex is removed from the impasse set and is placed into a random color class, moving neighbors of the vertex into the impasse set. Moves are selected so as to reduce the average degree of vertices in the impasse set; if several such moves are possible, one is chosen at random. With some small probability, a disimprovement is allowed, i.e. a move which increases the average degree of nodes in the impasse set. The probability of disimprovement is controlled by a *temperature* parameter. This allows the algorithm to avoid being trapped at local minima.

The S-Impasse algorithm also incorporates s-chain moves to keep the neighborhood of moves large. An s-chain consists of a tuple $(v, V_0, \ldots, V_s)$ where v is a member of the set V_0 and each set V_i is a subset of a distinct color class. The sets V_i have the property that placing set V_i into the color class $V_{(i+1) \bmod s}$ maintains a valid coloring. That is, the s-chain "shuffles" the coloring. (This is a generalization of Kempe chains, an idea used by Kempe [16] in his flawed proof of the four-color theorem.) A parameter, μ, is a scaling factor that determines how often the s-chain moves are performed. Morgenstern's version of the algorithm ran for a number of *iterations* before halting. Our algorithm runs until it has reached its *time bound.*

We set our parameters to be the same for all graph classes. The initial target is simply set to be the number of nodes in the graph. For explanations of exactly how the other parameters are used, see [20]. In our implementation, the length s of a Kempe chain is set to 3; the temperature is 0.6 and the parameter μ is set to 10. This time bound on the algorithm was set to three hours.

The Parallel S-Impasse Algorithm. In the parallel S-Impasse algorithm, several processors are run independently, each finding an initial coloring independently and then setting the initial target to be one less than the number of colors in the best coloring found. The processors then independently explore improvements starting from their initial coloring. When a processor finds a new best coloring, the new bound is broadcast to the other processors and all processors lower the size of the target coloring, moving a color class into the impasse set if necessary. When all processors have completed their total number of iterations, the algorithm stops. Previous studies of simulated annealing algorithms for other applications indicate that if the number of processors is small, this is a reasonable approach to parallelizing the S-Impasse algorithm, because the more paths that are explored independently, the smaller the expected time for one to reach a new best coloring. (See Azencott [2], for example.) We later show that this is in fact true for the S-Impasse algorithm on our test data. Another advantage of this parallel approach is that, since the computation of distinct processors is almost completely independent, the amount of interprocessor communication is kept to a minimum.

Processors do occasionally communicate. When a new coloring is found, a limited number (five of thirty-two in our experiments) of processors whose last target met is three greater than the new bound may abandon their partial solution and receive the current solution. They then do an s-chain move to avoid completely following the search of the sending processor. (Other processors who similarly appear to be following a dead end in their search but do not receive the current solution, do a large s-chain move instead.)

The Parallel Exhaustive Search Algorithm. This algorithm is a straightforward branch-and-bound procedure, as described by Johnson et al. [13]. A tree of partial colorings is expanded, using the size of the current best coloring to prune the tree. Each node in the tree represents a partial coloring and its children are all the possible extensions of that coloring obtained by coloring one more node. In the parallel implementation, a polling mechanism is used by idle processors to obtain work from busy processors. When a processor finds a new best coloring, the number of colors is broadcast to all of the other processors.

The Parallel Hybrid Algorithm. In this algorithm, a manager process starts a group of processors on parallel Exhaustive Search and another group on parallel S-Impasse. The manager maintains the current best coloring, and whenever either algorithm finds a new best coloring, this is sent to the manager, which broadcasts it to the other algorithm. Hybrid halts when one of the algorithms finishes its computation. Thus, the upper bound on the running time is the maximum of the time bound given to the S-Impasse algorithm and the completion of the exhaustive search.

We note that an execution of either the parallel S-Impasse or Exhaustive Search algorithms cannot be serialized to give an execution of the corresponding sequential algorithms. In particular, the parallel S-Impasse algorithm follows several paths in the search space, whereas the sequential algorithm follows just one path. Similarly, the order in which nodes of the search tree are expanded in the parallel Exhaustive-Search algorithm may be quite different than in the sequential algorithm.

3 Test Data

Our algorithms have been tested on random graphs, Leighton graphs, register allocation graphs and two graphs constructed from a course scheduling problem. Below we give a brief description of each graph class.

Leighton graphs. Leighton graphs [18] are random graphs with a fixed number of edges and predetermined chromatic number. The graphs are constructed by implanting cliques of sizes ranging from $\chi(G)$ to 2 into the graph. The nodes in the cliques are chosen in a manner that guarantees the chromatic number will not be larger than the pre-specified value. The density of these graphs is always less than 0.25, which Leighton claims is commonly the case in applications such as exam scheduling. (Our real data also fits this assumption.) We test our algorithm on Leighton graphs with 450 nodes and with chromatic number 5, 15 and 25.

Register Allocation Graphs. Register allocation graphs are used in compilers to determine a mapping of variables to registers. Variables that are active in the same range of code cannot be placed in the same register. A conflict graph is constructed, with variables as vertices and an edge representing variables live in the same range of code. Coloring this graph yields a mapping of variables to registers. If more colors are needed than there are registers, not all variables can be placed in registers. In this case, spill code must be inserted to remove some variables from the registers. Spill code removes some edges from the conflict graph; this subgraph can then be colored to see if the variables can be mapped to the registers with the spill code.

Preston Briggs of Rice University has constructed a system to test register allocation schemes. He has provided us with many program files along with code to output the original conflict graphs and several subgraphs constructed as spill code is inserted. The conflict graphs range in size from a couple of hundred vertices to several thousand vertices. In this study we consider conflict graphs ranging in size from around 200 vertices to around 850 vertices. Specifically, we study fourteen graphs from four different base programs, **mulsol**, **zeroin**, **fpsol2** and **inithx**. These graphs were constructed for a compiler having thirty-two registers available. There are five mulsol files and three each of the zeroin, fpsol2 and inithx files, resulting from the initial graph and the graphs formed from spill code insertion. In each case, the register allocation scheme used by Briggs only successfully colored (i.e. using thirty-two colors or less) the last graph in the set.

$G_{n,p}$ graphs. Commonly used in testing graph coloring algorithms, a $G_{n,p}$ graph has n vertices, and an edge between each pair of vertices with independent probability p. We test our algorithms on $G_{n,0.5}$ graphs for n = 70, 125, 250, 500 and 1,000.

Class Scheduling Graphs. Many high schools ask students to select a set of desired courses for the coming year and then attempt to construct a timetable that allows all students to take their chosen courses. A timetable with no conflicts corresponds to a coloring of a graph with vertices corresponding to courses and edges between two vertices if a student has requested both courses or the same teacher teaches both courses.

We have obtained the scheduling data from a local high school that has approximately 500 students. There are seven periods in a school day and the entire year (two semesters) is scheduled at one time. Counting each section of a course separately, there are 385 vertices in the class graph. Since we are working from a final schedule, the graph is guaranteed to be 14-colorable.

For this study we have looked at algorithm performance on the entire graph, called **school.dat**, as well as the subgraph corresponding to removing all references to study halls, called **school-nsh.dat**.

Generated Class Scheduling Graphs. We have begun to work on a generator for graphs modeling the class graphs used in this study. We started by examining the real data to get an idea of the important factors affecting the composition of the class graph.

Since we have only one data set we cannot be sure of all the factors, nonetheless we believe the following factors are important in the composition of a class graph. The first important factor is division of students into separate groups, such as different grades and/or different tracks of study. These divisions are important because each of these groups will have a core set of classes taken only by students in this group – and the vertices corresponding to classes taken by different groups will be independent. The second important factor is the selection of courses that intersect the interests of various groups; for example music courses will be shared between all grades. The probability of students in a group taking courses that also interest another group will vary among the groups. This variance will affect the number of edges between the courses from these groups. Currently, our generator concentrates on handling these two factors.

The assignment of sections after selecting courses must also be done when generating a class graph. We do this very simplistically, setting a bound on the number of students in a given section of course, then assigning as many sections as necessary.

The generator constructs a random schedule for each student, based on the parameters. We then add sections and process the data in the same manner as the real data.

As a basis for comparison, we compare our results on a generated graph, **model-1.dat** with a $G_{n,p}$ graph of the same number of nodes and density, $G_{299,0.23}$, and with a graph based on the real data but with section information added by the generator, **School-as.dat**.

Geometric Graphs. Geometric random graphs, $Rx.y$, are formed by randomly placing x vertices in a unit square, then putting edges between any two vertices which are within $.y$ of each other. These graphs may model applications such as assigning cellular phone frequencies. We test our algorithms on 8 instances of geometric graphs, of size 125, 250, 500, and 1000, with y parameter of $1, 5$. We also test them on complements of $Rx.1$ graphs.

Latin Square Graph. This graph represents a problem from design theory, relating to latin squares. The graph is a 900 vertex graph, with independent sets no larger than 10 vertices. It is an open problem whether or not this graph can be colored in 90 colors.

k-Partite Graphs. These graphs are $G_{n,p,k}$ graphs modified to have the density of a $G_{n,p}$ graph. We look at graphs of size 300 with $k = 20, 26, 28$ and of size 1000 with $k = 50, 60, 76$. The target p is 0.5.

Graphs for Parallelizing Iterative Solutions of Sparse Linear Systems. Solving large sparse linear systems can be done iteratively in parallel using using the Gauss-Seidel iteration method, in which independent components of the system are updated in parallel. Treating the matrix representing the system as an adjacency matrix, with positive entries representing edges, the coloring of the graph with k colors reveals a decomposition of an iteration into k steps. At step i of an iteration, all the components in color class i are updated in parallel. The parallelization does not actually require a true coloring of the graph; it is sufficient to color the vertices so that no positive cycle contains only vertices of the same color. There are many applications of sparse linear systems; our graphs are examples from power systems, of sizes 1993, 1084, 707, and 147.

Final Exam Scheduling. As with class scheduling, final exam scheduling must avoid scheduling courses taken by the same student simultaneously. Vertices are courses, edges are between courses taken by the same student. We have acquired data from Florida Institute of Technology (provided by Lynn Kiaer) for exam scheduling. Each edge has an integer weight between one and three, representing the severity of the conflict. To schedule the exams, the graph must be colored with six or fewer colors – this may require leaving conflicts in the coloring. The goal of the problem is to have as few as possible conflicts of the highest weights. We examine three graphs from this data; graph **fl-tech.1** contains all the edges, **fl-tech.2** removes edges of least severity, and **fl-tech.3** contains ony the edges of highest weight.

4 Coloring Results

In this section, we discuss the performance of the Hybrid algorithm on the different test graph types listed in Section 3. For each class of graphs, we compared Hybrid with the simpler RLF and Saturation heuristics, and also compared the quality of our colorings with those obtained in previous work. Briefly, our main conclusions are as follows.

- Overall, the Hybrid algorithm performs very well, easily surpassing previous experimental results for untuned algorithms. Moreover, Hybrid produces good colorings on a wider range of graphs than any previously reported algorithm. This is perhaps not too surprising, since we are using a powerful machine, and combine more than one previously touted technique. Still, our results are obtained with no tuning of parameters; this is an important advantage over previous work. Our algorithm performs most poorly on large, randomly generated graphs. For example, using his tuned S-impasse algorithm, Morgenstern obtains better results on large $G_{n,p}$ graphs. Also, Johnson et al. obtain better colorings with their XRLF algorithm. A detailed comparison of Hybrid with other algorithms on different classes of graphs is given in Section 4.1.

- Of the five types of test data, three are easily colored even by the simple RLF and Saturation heuristics; one is optimally colored by Hybrid but not by the simple heuristics, and one appears to be very hard. However, we stress that just because in several cases the test graphs are easy to color, does *not* mean that the applications are easily solved. In several cases, such as the class and exam scheduling graphs and the sparse matrix graphs, finding an optimal coloring is not sufficient to solve the problem at hand. Instead, colorings satisfying additional restrictions are really needed. Adapting good coloring heuristics to really solve these problems is an important direction for further research. We discuss this further in Section 4.2.

- The Hybrid algorithm parallelizes well. One reason is because the number of iterations needed by S-impasse decreases as the number of processors increase, supporting our approach of independently exploring several coloring modifications independently. Also, in the exhaustive search algorithm the work involved in expanding the search tree is effectively shared among the processors. We present some experimental data supporting this in Section 4.3.

- There is often quite a variance in the running time depending on the random seed. The variance exists even when disregarding runs in which the size of the coloring differed. Using only those runs finding the most common color size, Table 1 summarizes the average running time and variance for the Hybrid algorithm on several graphs. We also examined repeated runs of Hybrid using the same random seed, to see if machine effects cause the variance, but found very little difference in running time while using the same seed.

4.1 Evaluation of Randomly Generated Data

All of our algorithms were run on Thinking Machine's CM-5 machine. RLF and Saturation were run on 1 processor, while Hybrid was run on 32 processors, which were partitioned with one acting as manager, 8 executing the parallel Exhaustive-Search algorithm, and 23 executing the parallel S-Impasse algorithm. Each computation was stopped after three hours; we report the time less than that if the best coloring was found earlier. A nice feature of our Hybrid algorithm is that if the Exhaustive-Search procedure is completed, the optimal coloring is known. Thus, we are able to report optimal coloring sizes for several graphs. All of these results are summarized in Appendix I and Figure 1. In what follows, we discuss each graph class in turn.

Leighton Graphs (see Table 2). Hybrid outperformed the simpler RLF and Saturation heuristics on all Leighton graphs. Hybrid found optimal colorings on all of the 5-colorable graphs, two of the four 15-colorable graphs and two of the four 25-colorable graphs. Close to optimal colorings (off by one or two) were found for the remaining graphs. The time required to find the best colorings ranged from less than half a minute (on a 25-colorable graph) to almost three hours (on a 15-colorable graph).

In the execution of the Hybrid algorithm on the 5-colorable Leighton graphs, the Exhaustive-Search and parallel S-Impasse alternated regularly in decreasing the current best number of colors, finishing with the Exhaustive-Search. On the other Leighton graphs, with chromatic numbers 15 and 25, the Exhaustive-Search algorithm was useful in decreasing the current best coloring for several colors in the initial stages, and to end the computation if the chromatic number was found. Progress after the initial stages, however, was made by the S-Impasse algorithm.

Hybrid also surpassed previous results of Morgenstern. His tuned, sequential S-Impasse algorithm, only found a 20-coloring, in somewhat less than an hour, for each of the two 15-colorable graphs on which we find a 16-coloring.

Random $G_{n,p}$ Graphs (see Table 3). The Hybrid algorithm fails to come close to the lower bound estimates on the large $G_{n,p}$ graphs. On the $G_{500,0.5}$ graph, which has an estimated lower bound of 46, the best coloring found was 52 colors, found by Hybrid in about 2.19 hours. On the $G_{1000,0.5}$ graph, which has an estimated lower bound of 80, the best coloring found was 99 colors, found in just under 2.5 hours. Still, Hybrid always outperformed the simpler heuristic algorithms on all of these graphs.

The poor results are perhaps not too surprising, as Morgenstern also reported difficulty in obtaining good colorings for large $G_{n,p}$ graphs, even with his tuned sequential S-Impasse algorithm. For example, he reports running times of about 65 hours to find a 90-coloring of a $G_{1000,0.5}$ graph, and 40 hours to find a 50-coloring of a $G_{500,0.5}$ graph (on a VAX 11/780).

Geometric Graphs (see Table 4). On six of the twelve geometric graphs, we were able to find optimal colorings using Hybrid. However, on two of the 1,000-node graphs, we have no estimates on the optimal coloring size, and we do not know how close our colorings are to the optimal colorings.

Previous results were reported by Johnson et al. for geometric graphs of size 500, so we compare our results with theirs on these graphs. The performance of Hybrid is mixed. On the graph with an edge between two points if the distance between them is less than .1, 12 colors is optimal, and Hybrid obtains a 12-coloring in less than half a minute (and most of this time is actually spent reading the graph). On the graph with an edge between two points if the distance between them is greater than $>$ .9, Hybrid obtains an 85 coloring in three out of four runs, in an average time of about 2 hours. In contrast, the best coloring obtained by the RLF and Saturation algorithms were of size 88 and 89, respectively. Also, Johnson et al. reported that the best coloring they obtained was of size 85, using a tuned version of an annealing-type algorithm (the Fixed-K Annealing algorithm) which ran for over 75 hours on a Sequent. On the third graph, with an edge between two points if the distance between them is less than .5, Hybrid obtains a coloring of size 128 in about 1.5 minutes, but fails to find a better coloring in 3 hours. Johnson et al. report that in several runs of the Saturation algorithm, a coloring of size 124 was obtained, so Hybrid fails to match this coloring.

k-Partite Graphs (see Table 5).

Again, the Hybrid does not come close to the optimal colorings on these graphs, although it obtains better colorings than RLF and Saturation. When given an initial target corresponding to k, instead of having Hybrid start from an initial coloring and steadily decrease the number of colors, Hybrid successfully found the optimal coloring of two of the 300 vertex graphs.

4.2 Evaluation of Application Data

We have gathered graphs from several applications for the purposes of studying how well common heuristics perform on them, whether or not it is feasible to use an exact coloring algorithm on graphs from applications, and to make them more widely available for benchmarking and testing coloring algorithms.

Class Scheduling Graphs (see Table 6). For both class scheduling graphs, an optimal 14-coloring was found both by parallel S-Impasse and Hybrid. Surprisingly, for the School.dat graph, the Exhaustive-Search algorithm was the main workhorse in Hybrid. In fact, the parallel S-Impasse algorithm alone takes 13 minutes to find the fourteen-coloring (compared to an average of 46.2 seconds for they Hybrid algorithm). The sequential S-Impasse algorithm only finds a 23-coloring for the School.dat graph, indicating the advantage of parallelism. The Saturation algorithm finds an 18-coloring of the School.dat graph, 8 colors better than the RLF algorithm.

The School-nsh.dat graph is more difficult to color than the School.dat graph, even though School-nsh.dat is a subgraph of School.dat. Hybrid takes 66.4 seconds to find an optimal coloring, with exhaustive search finding several early colorings, then S-impasse taking over the work until the last few colorings, including the optimal, which are found by exhaustive search. The parallel S-Impasse algorithm takes longer, 11.5 minutes, to find the optimal coloring. Sequential S-Impasse takes 40 minutes to find an optimal coloring of the School-nsh.dat graph. RLF does better than Saturation in the School-nsh.dat graph, needing 22 colors compared to Saturation's 28 colors.

Even though the graphs are over 300 vertices and of density around 0.25, exact coloring is quite feasible. The Hybrid algorithm completed its exhaustive search on the School.dat graph in an average of 78 seconds, and on the School-nsh.dat graph in an average of 90 seconds.

Although coloring the school graphs is relatively easy for our Hybrid heuristic, the main difficulty of scheduling courses remains. The assignment of sections greatly affects the chromatic number, as we see below in our comparison of the generated class graph to the real data with section numbers added automatically.

Generated Class Scheduling Graphs (see Table 7). We find that the results on the generated graph appear to be more similar to the results on the real data with sections added by the generator than to the $G_{n,p}$ graph of the same density. In particular, the $G_{n,p}$ graph needs fewer colors.

School-as.dat is colored in 28 colors by both the RLF and Saturation algorithms. The Hybrid and parallel S-Impasse each manage to find a 28 coloring of the graph.

The graph model-1.dat, built by our generator based on parameters similar to the real data, was colored by RLF in 32 colors, and Saturation in 31 colors. The Hybrid and parallel S-Impasse algorithm each found a 28 coloring of the graph.

The density of both School-as.dat and model-1.dat was around 0.23. We built a random $G_{299,0.23}$ graph, having the same number of nodes and the same density as model-1.dat. This graph was colored with far fewer colors by RLF and Saturation, 19 and 22 respectively. Hybrid and S-Impasse both found a 17 coloring.

Our conclusion is that our generator has constructed a graph that is a better model for testing heuristics on class graphs than a $G_{n,p}$ graph. We note, however, that model graphs do not yet appear to capture all aspects of the real data. The largest clique found by dfmax (a semi-exhaustive greedy clique finding algorithm of Matula and Johri) in the model graph is 25 while the school-as.dat graph has a clique of size 18. We are doing further work both theoretically and experimentally in this area.

Unlike the 14-colorable school graphs, the model graph and the school graph with sections generated automatically are not easily exactly-colored. Our Hybrid algorithm spent three hours on each graph without finishing its search.

The difficulty of coloring School-as.dat suggests that the assignment of sections in such a way to keep the chromatic number low is a valuable area of study.

Register Allocation Graphs (see Table 8). The Saturation and RLF performed just as well as the Hybrid and parallel S-Impasse on these graphs, and all algorithms halted quickly. We verified that the results were close to optimal by finding lower bounds on the chromatic number of the register allocation graphs (again given in parentheses) by running our exhaustive search algorithm on subsets of the vertices. For most of these graphs, the lower bound we found was identical to the number of colors used. David Johnson confirmed that the rest of the colorings were also optimal by finding cliques of the coloring size. S-Impasse found the same coloring as Hybrid, so we do not report it in the table.

Our Hybrid algorithm did not prove that it exactly colored the register allocation graphs. However, adding a simple clique finder, such as Matula and Johri's dfmax, to the algorithm to find lower bounds would have enabled the algorithm to quickly prove the colorings were optimal.

Graphs for Parallelizing Iterative Solutions of Sparse Linear Systems (See Table 9). The RLF algorithms optimally colored all of the graphs. The Saturation algorithm exactly covered three of the four graphs and used four instead of three colors to color s1084.dat. Two of the graphs, s147.dat and s1084.dat were quickly exactly colored by the sequential exhaustive search algorithm. Lower bounds on the other two graphs were found quickly by dividing the graphs into 50 vertex sections and coloring with exhaustive search. We did not run the parallel heuristic on these graphs because the sequential algorithms worked so quickly on them.

While s147.dat and s1084.dat were easily colored with exhaustive search, the s1993.dat graph was too large for our program and the search was unsuccessful on the s707.dat graph.

The colorings of these graphs are used to parallelize the computation of a linear sparse system by computing values of the elements in each color class in parallel. Except for the s707.dat graph, all the colorings had a very low number of colors, indicating few parallel steps per iteration. The number of steps actually needed may be lower, even though our algorithms find the exact coloring, because the parallelization of the linear computation does not need a strict coloring. It is sufficient that the graph is colored so that there exists no positive cycle in the graph with all nodes in it colored the same color. Thus, although these graphs are easy to color, the algorithms are not quite solving the problem posed by the application.

Final Exam Scheduling Graphs (See Table 10). Graphs fl-tech.1 and fl-tech.2 were quickly and exactly colored using the Hybrid heuristic. The third graph, fl-tech.3 was quickly colored with 6 colors, but Hybrid did not show it to be exactly colored. Both RLF and Saturation found the minimum coloring on each graph. Only fl-tech.3 was colored with the six colors needed to actually effectively schedule the exams. Kiaer has constructed heuristics to use the weights of the conflicts to find a 6 coloring with no severe (weight 3) conflicts, 5 medium (weight 2) conflicts and 42 small (weight 1) conflicts [17].

We see again with this application that coloring the graph itself is not difficult, but modifications are needed to graph coloring in order to actually solve the problem posed by the application.

Latin Square Graph. All the algorithms perform poorly on this application. The RLF algorithm uses at least 146 colors to color this graph. The Saturation algorithm uses 132 colors. As in the course scheduling graphs, this is a reversal of the typical performance of RLF and Saturation on random graphs. The Hybrid algorithm can get no fewer than 109 colors on this graph. We conclude that this graph is a hard test for graph coloring heuristics.

Exact coloring appears completely infeasible at this point for the Latin Square graphs.

Our study of these application graphs shows that several of the applications (register allocation, matrices for sparse linear systems, exam scheduling) provide graphs which are quite easy to color. Course scheduling is a little harder, with RLF and Saturation being insufficient to color the graphs but being easily colored by the Hybrid heuristic. One of the applications, Latin Square, is very difficult. We conjecture that many applications will fall into the easy or moderately difficult category, corresponding to our observations in this study.

Our study of applications also shows that unlike random graphs, where RLF often tops the performance of Saturation, there is no clear winner when comparing the two on applications. For register allocation, exam scheduling, and most sparse matrices, they gave the same results; RLF was better on one sparse matrix and one course graph, while Saturation was better on the other course graph and the latin square graph.

We also note that coloring the graphs of many of these applications, namely course scheduling, final exam scheduling, and sparse matrices, does not actually solve the problem originally given in the application. Thus, although these graphs do not provide a great challenge to current heuristics for coloring, they do offer a challenge to modify heuristics or add additional algorithm techniques in order to better solve the exact problem posed by these applications.

4.3 Advantages of Parallelism

Clearly, the Hybrid algorithm must always produce colorings that are as least as good as the sequential S-Impasse algorithm. Our hopes in undertaking this project were that Hybrid would produce good colorings in significantly less time than the sequential S-Impasse algorithm. We were also curious if the exhaustive search component of Hybrid would be useful. In order to understand the advantages of a parallel implementation, we compared the parallel and sequential versions of the S-impasse and Exhaustive-Search algorithms separately.

Parallel vs. Sequential S-Impasse. Our parallel version of S-Impasse finds good colorings more quickly than the sequential version. For several graphs, we counted the number of iterations needed to find the best coloring, in the parallel and sequential algorithms, and found that the number of parallel iterations was consistently much less in the parallel implementation (see Table 12). A typical example is the 5-colorable Leighton graph, where four times as many iterations were needed by the sequential algorithm than by the parallel algorithm, to find a 7-coloring, and a 6-coloring was never found by the sequential algorithm. The difference in number of iterations needed to reach the new coloring is reflected in the running times: the sequential algorithm required 58 minutes to find the 7-coloring, whereas the parallel algorithm found this in 11 minutes. Thus, our approach to parallelizing the S-Impasse is sound. Further testing is needed to see a more general correlation between the improvement in performance and the number of processors.

Parallel vs. Sequential Exhaustive Search. We ran several additional experiments to test the parallel Exhaustive-Search algorithm, and conclude that it performs very well. Table 13 shows the running time of parallel Exhaustive-Search on a 70-node random graph, with the total number of nodes expanded, and the minimum and maximum number expanded by each processor. While the 70-node graphs are not big enough to give all thirty-two CM-5 processors work at all times, a speedup of a factor of 3.2 is observed for four processors and more speedup is observed as processors increase. Also encouraging is that the number of search nodes examined in the parallel implementation is rarely more than five percent more than the total number of nodes examined the sequential implementation. In other studies of several graphs, including the Leighton graphs, we found that the number of nodes expanded by the processors are generally within ten percent of each other.

Hybrid vs. Parallel S-Impasse. On the parallel environment on the CM-5, it was very easy to experiment with the Hybrid algorithm, once the parallel Exhaustive-Search and S-Impasse algorithms were implemented. The performance of the two algorithms was similar, with Hybrid doing slightly better overall. The Exhaustive-Search component of Hybrid was most useful on graphs with low chromatic number.

5 Conclusions and Future Work

We were pleased to find that the Hybrid algorithm performed very efficiently, and consistently produced the best colorings. Based on a comparison with Morgenstern's previous results, we conclude that tuning of the algorithms should be useful in improving our results on the large random $G_{n,p}$ graphs, but on all other graph classes, our untuned algorithm matches or exceeds previously reported best results.

In further work, we plan to add XRLF to our Hybrid, to run in parallel with S-Impasse and Exhaustive-Search. In contrast to S-Impasse, the XRLF algorithm runs well on random $G_{n,p}$ graphs, given enough time; for example, an 86-coloring of a $G_{1000,0.5}$ graph is found in 68.3 hours [13]. (This was on a VAX 750, which is 20-100 times slower than current machines.) We already have a working parallel implementation of XRLF, but unfortunately, due to lack of time and competition for the CM-5, we have not yet been able to test this hybrid. In our implementation, we use Hybrid as a replacement for the exhaustive search alone at the end of the XRLF algorithm.

Our experience with the class scheduling graphs strongly suggests that more effort should be made to find applications and real data to compare algorithms. Furthermore, although coloring is important in these applications, it is clear that in many cases that the problem is more complex than simply finding good colorings. It is still unclear whether good coloring heuristics can really be applied in these applications.

6 Acknowledgements

We thank Robert Mills, Lynn Kiaer, Preston Briggs, Monika ten Bruggencate and Wendy Myrvold for their helpful discussions about applications and help in acquiring data, and Alex Chan for his work entering the class scheduling data.

References

[1] Appel, K. I., W. Haken and J. Koch. Every planar map is four colorable, Part I: Discharging, *Illinois Journal of Mathematics*, 21, 1977, 429- 490.

[2] R. Azencott, Editor. *Simulated Annealing: Parallelization Techniques*, New York, John Wiley and Sons, 1992.

[3] B. Bollobas and A. Thomason. Random Graphs of Small Order. *Ann. Discrete Math.* **28** 1985, 47-97.

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

[5] G.J. Chaitin and M. Auslander and A.K. Chandra and J. Cocke and M.E. Hopkins and P. Markstein, Register Allocation via Coloring, *Computer Languages*, 6, 1981, 47–57.

[6] G.J. Chaitin, Register Allocation and Spilling via Graph Coloring, *Proceedings of the ACM SIGPLAN 82 Symposium on Compiler Construction*, 1982, 98–105.

[7] D. de Werra. An introduction to timetabling, *European Journal of Operations Research*, 19, 1985, 151–162.

[8] M. R. Garey and D. S. Johnson and H. C. So. An application of graph coloring to printed circuit testing, *IEEE Transactions on Circuits and Systems*, 23, 1976, 591–599.

[9] J.W. Greene and K. J. Supowit. Simulated annealing without rejected moves, *IEEE Transactions on Computer-aided Design*, vol CAD-5, 1, January 1986, 221-228.

[10] G.R. Grimmet and C.J.H. McDiarmid. On colouring random graphs, *Mathematical Proceedings of the Cambridge Philosophical Society*, 77, 1975, 313–324.

[11] F. Glover. Tabu search, part 1, *ORSA Journal on Computing*, 1, 1989, 190–206.

[12] M. M. Halldórsson. A still better performance guarantee for approximate graph coloring, DIMACS Technical report 1990, 91–35.

[13] D. S. Johnson, C. R. Aragon, L. A. McGeoch and C. Schevon. Optimization by simulated annealing: an experimental evaluation; part II, graph coloring and number partitioning, *Operations Research*, 3, 1991, 378–406.

[14] A. Johri and D. W. Matula. Probabilistic bounds and heuristic algorithms for coloring large random graphs, Technical report, Southern Methodist University, Texas, 1982.

[15] R. M. Karp. Reducibility among combinatorial problems, in R.E. Miller and J.W. Thatcher (ed.), *Complexity of computer computations*, Plenum Press, New York, 1972, 85–103.

[16] A. B. Kempe, On the geographical problem of the four-colors, *American Journal of Mathematics*, 2, 1879, 193–200.

[17] Kiaer, Lynn. Discrete Optimization Strategies for Timetabling, Ph.D. Dissertation, Department of Applied Mathematics, Florida Institute of Technology, June 1992.

[18] F.T. Leighton. A graph coloring algorithm for large scheduling problems, *Journal of Research of the National Bureau of Standards*, 84, 1979, 489–506.

[19] C. Lund and M. Yannakakis. On the hardness of approximating minimization problems, *Proceedings 25th ACM Symposium on Theory of Computing*, 1993, 286-293.

[20] C. A. Morgenstern. Algorithms for general graph coloring, Doctoral Dissertation, Technical report CS89-16, Department of Computer Science, University of New Mexico, Albuquerque, 1989.

[21] R.J. Opsut and Fred S. Roberts. On the fleet maintenance, mobile radio frequency, task assignment and traffic phasing problems, in G. Chartrand, Y. Alavi, D.L. Goldsmith, L. Lesniak–Foster and D.R. Lick, *The Theory and Applications of Graphs*, John Wiley & Sons, New York, 1981, 479–492.

[22] E. L. Poole and J. M. Ortega, Multicolor ICCG methods for vector computers, *SIAM Journal of Numerical Analysis*, 24, 6, 1987, 1394–1418.

[23] D. C. Wood. A technique for coloring a graph applicable to large scale time-tabling problems, *Computer Journal*, 12, 1969, 317–319.

Appendix I

Figure 1: Comparison of Hybrid Colorings with best known sequential colorings. Difference between number of colors used by Hybrid and best coloring known is plotted. (Points are the number of colors used by Hybrid.) Points above 0 represent graphs in which Hybrid obtained better colorings.

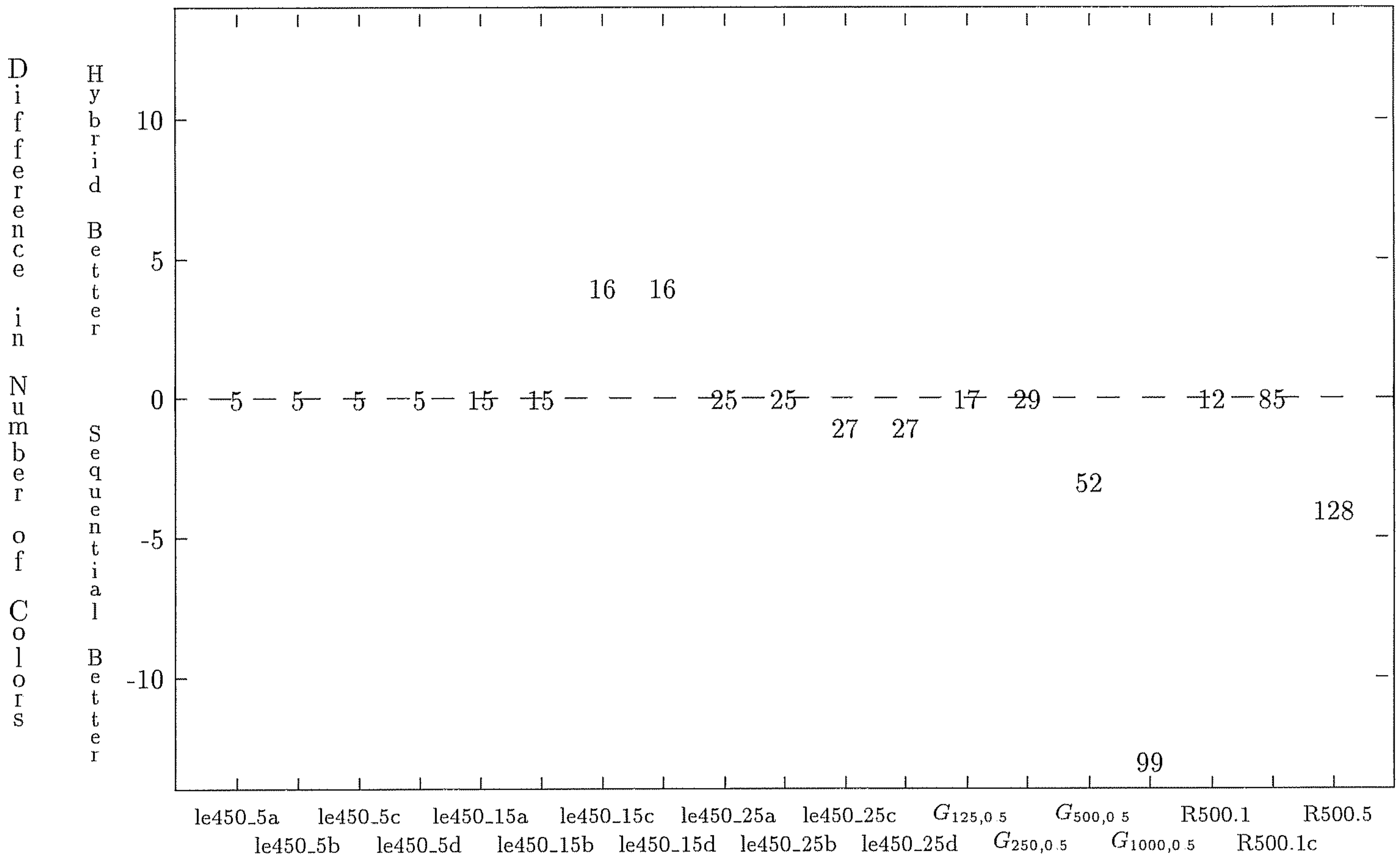

Table 1: Running time and standard deviations for several graphs. The running times for the coloring result most commonly achieved in 5 trials were used to calculate the variance. (This is not the best coloring in all cases.) Time is in seconds.

Graph	Runs	Colors	Running Times			
			Min	Average	Std. Deviation	Max
$G_{125,0.5}$	5	17	1199.22	4043.63	2508.8	8037.7
$G_{250,0.5}$	4	29	3612.74	5371.12	1879.61	8013.02
$G_{500,0.5}$.col	3	53	3343.72	4959.88	1851.77	6980.43
$G_{1000,0.5}$	3	100	4631.97	4755.06	179.242	4960.7
R125.1	5	5	50	64.6	22.2666	104
R125.1c	5	46	60	85	22.6716	120
R125.5	5	37	31.47	32.986	1.98744	36.38
R250.1	5	8	22	22	0	22
R250.1c	5	64	110.7	278.16	160.948	505.8
R250.5	5	66	38.9	39.88	0.563028	40.3
DSJR500.1	5	12	24.5	26.64	4.11983	34
DSJR500.1c	3	85	4717.2	7246.53	2729.44	10139.6
DSJR500.5	5	128	85.6	90.5	5.245	96.2
R1000.1	5	20	49.4	49.88	0.268328	50
R1000.1c	2	103	324.9	326	1.55563	327.1
R1000.5	3	246	210	213.667	3.51188	217
flat300_20	5	20	236.8	274.3	36.6314	329.3
flat300_26	3	32	6904.5	9109.83	1934.17	10518.1
flat300_28	5	33	454.6	1913.54	1515.12	3786.9
flat1000_50	3	97	7172.8	7482.27	364.67	7884.3
flat1000_60	5	97.8	3766	6288.36	1469.67	7518.7
flat1000_76	4	99	5697	6497.85	664.774	7100.3
latin_square	3	109	5266.4	6874.9	1663.54	8588.5
le450_15a	5	15	88	162.62	75.522	278.1
le450_15b	5	15	113.3	178.36	45.0573	226.1
le450_15c	5	16.6	1016.1	2229.61	1114.42	3828.8
le450_15d	5	16.8	1303.5	2859.6	1999.92	5754.8
mulsol.1	5	49	27	27.2	0.447214	28
school1.dat	5	14	37.6	46.26	7.82292	55.2
school1_nsh.dat	5	14	54.2	66.4	9.36616	76.4

Table 2: Leighton Graphs with 450 vertices. In the leftmost column, graphs are described by file name (as in Challenge files), with chromatic number in parentheses. For each of RLF, Saturation and Hybrid, the following data is listed: (i) The size of the best coloring found by each algorithm. (ii) The number of runs achieving this coloring, over the total number of runs. (iii) The average running time, over all runs obtaining the best coloring.

Graph	RLF			Saturation			Hybrid(32 Procs)		
	(i)	(ii)	(iii)	(i)	(ii)	(iii)	(i)	(ii)	(iii)
le450-5a(5)	8	(1/1)	(26.23 secs)	12	(1/1)	(47.64 secs)	5	(1/1)	(15.11 mins)
le450-5b(5)	7	(1/1)	(26.12 secs)	11	(1/1)	(38.45 secs)	5	(1/1)	(7.66 mins)
le450-5c(5)	5	(1/1)	(28.07 secs)	13	(1.1)	(37.99 secs)	5	(1/1)	(28.07 secs)
le450-5d(5)	8	(1/1)	(28.01 secs)	13	(1/1)	(41.51 secs)	5	(1/1)	(6.58 mins)
le450-15a(15)	17	(5/5)	(27.20 secs)	17	(5/5)	(41.00 secs)	15	(5/5)	(2.71 mins)
le450-15b(15)	17	(5/5)	(28.20 secs)	17	(5/5)	(41.00 secs)	15	(5/5)	(2.97 mins)
le450-15c(15)	24	(4/5)	(35.00 secs)	24	(5/5)	(32.50 secs)	16	(2/5)	(44.42 mins)
le450-15d(15)	24	(3/5)	(33.00 secs)	24	(1/5)	(43.00 secs)	16	(1/5)	(1.60 hrs)
le450-25a(25)	25	(1/1)	(27.80 secs)	25	(1/1)	(28.80 secs)	25	(1/1)	(27.8 secs)
le450-25b(25)	25	(1/1)	(27.96 secs)	25	(1/1)	(31.00 secs)	25	(1/1)	(27.96 secs)
le450-25c(25)	28	(1/1)	(32.06 secs)	30	(1/1)	(45.60 secs)	27	(1/1)	(2.92 mins)
le450-25d(25)	28	(1/1)	(31.99 secs)	31	(1/1)	(41.09 secs)	27	(1/1)	(64.00 secs)

Table 3: Random $G_{n,p}$ Graphs. In the leftmost column, graphs are described by node size and probability p, with an estimated lower bound on the chromatic number in parentheses. For each of RLF, Saturation and Hybrid, the following data is listed: (i) The size of the best coloring found by each algorithm. (ii) The number of runs achieving this coloring, over the total number of runs. (iii) The average running time, over all runs obtaining the best coloring.

Graph	RLF			Saturation			Hybrid(32 Procs)		
	(i)	(ii)	(iii)	(i)	(ii)	(iii)	(i)	(ii)	(iii)
$G_{70,0.5}$(11)	14	(1/1)	(20 sec)	13	(1/1)	(33 secs)	11	(1/1)	(50.5 secs)
$G_{125,0.5}$(16)	22	(5/5)	(22.00 secs)	23	(5/5)	(35.80 secs)	17	(5/5)	(1.13 hrs)
$G_{250,0.5}$(27)	35	(3/5)	(29.60 secs)	37	(2/2)	(27.00 secs)	29	(4/5)	(1.49 hrs)
$G_{500,0.5}$(46)	62	(1/5)	(1.70 mins)	63	(5/5)	(52.00 secs)	52	(1/5)	(2.19 hrs)
$G_{1000,0.5}$(80)	112	(3/5)	(9.05 mins)	117	(5/5)	(1.33 mins)	99	(1/5)	(2.28 hrs)

Table 4: Geometric Graphs. In the leftmost column, graphs are described by file name (as in Challenge files). If the optimal coloring size is known, this is included in parentheses in the leftmost column. For each of RLF, Saturation and Hybrid, the following data is listed: (i) The size of the best coloring found by each algorithm. (ii) The number of runs achieving this coloring, over the total number of runs. (iii) The average running time, over all runs obtaining the best coloring.

Graph	RLF			Saturation			Hybrid(32 Procs)		
	(i)	(ii)	(iii)	(i)	(ii)	(iii)	(i)	(ii)	(iii)
R125.1(5)	5	(5/5)	(40 secs)	5	(5/5)	(28 secs)	5	(5/5)	(1 min)
R125.1c(46)	46	(5/5)	(2 mins)	46	(5/5)	(30 secs)	46	(5/5)	(1.4 mins)
R125.5	39	(5/5)	(20 secs)	38	(5/5)	(34 secs)	37	(5/5)	(32 secs)
R250.1(8)	8	(5/5)	(22 secs)	8	(5/5)	(29 secs)	8	(5/5)	(22 secs)
R250.1c(64)	65	(1/5)	(46 secs)	65	(5/5)	(39 secs)	64	(5/5)	(4.6 mins)
R250.5	70	(5/5)	(39 secs)	67	(5/5)	(36 secs)	66	(5/5)	(40 secs)
DSJR500.1(12)	12	(3/5)	(25 secs)	14	(5/5)	(38 secs)	12	(5/5)	(27 secs)
DSJR500.1c	89	(2/4)	(2.5 mins)	88	(5/5)	(86 secs)	85	(3/4)	(2 hrs)
DSJR500.5	132	(5/5)	(2 mins)	130	(5/5)	(53 secs)	128	(5/5)	(1.5 mins)
R1000.1(20)	20	(5/5)	(50 secs)	20	(5/5)	(61 secs)	20	(5/5)	(50 secs)
R1000.1c	104	(2/4)	(10 mins)	104	(5/5)	(4.3 mins)	101	(1/5)	(2.4 hrs)
R1000.5	261	(2/2)	(12 mins)	248	(5/5)	(2.6 mins)	243	(1/5)	(3.7 mins)

Table 5: k-partite Graphs. In the leftmost column, graphs are described by file name (as in Challenge files). The optimal number of colors is given in parentheses. For each of RLF, Saturation and Hybrid, the following data is listed: (i) The size of the best coloring found by each algorithm. (ii) The number of runs achieving this coloring, over the total number of runs. (iii) The average running time, over all runs obtaining the best coloring.

Graph	RLF			Saturation			Hybrid(32 Procs)		
	(i)	(ii)	(iii)	(i)	(ii)	(iii)	(i)	(ii)	(iii)
flat300-20(20)	39	(4/5)	(58 secs)	41	(5/5)	(52 secs)	20	(5/5)	(4.5 mins)
flat300-26(26)	40	(1/5)	(44 secs)	41	(5/5)	(39 secs)	32	(3/5)	(2.5 hrs)
flat300-28(28)	40	(3/5)	(43 secs)	43	(5/5)	(38 secs)	33	(5/5)	(32 mins)
flat1000-50(50)	109	(1/5)	(8.6 mins)	114	(5/5)	(2.3 mins)	96	(1/5)	(2.3 hrs)
flat1000-60(60)	110	(3/5)	(8.9 mins)	112	(5/5)	(2.5 mins)	97	(2/5)	(1.9 hrs)
flat1000-76(76)	113	(2/4)	(8.9 mins)	115	(5/5)	(2.5 mins)	99	(4/4)	(1.8 hrs)

Table 6: Class Scheduling Graphs. The first is a 385-node graph; the second is a subgraph of this of size 352. Both are 14-colorable. For each of RLF, Saturation and Hybrid, the following data is listed: (i) The size of the best coloring found by each algorithm. (ii) The number of runs achieving this coloring, over the total number of runs. (iii) The average running time, over all runs obtaining the best coloring.

Graph	RLF			Saturation			Hybrid(32 Procs)		
	(i)	(ii)	(iii)	(i)	(ii)	(iii)	(i)	(ii)	(iii)
School.dat (14)	26	(5/5)	(32.20 secs)	17	(5/5)	(33.00 secs)	14	(5/5)	(46.00 secs)
School-nsh.dat (14)	22	(5/5)	(30.80 secs)	28	(5/5)	(42.00 secs)	14	(5/5)	(66 secs)

Table 7: Generated Class Scheduling Graph (model-1.dat) compared with a random $G_{299,0.23}$ graph and a graph based on real course selections but section numbers added automatically (School-as.dat). The size of the coloring found by each algorithm is listed, with running times in parentheses for Hybrid and S-Impasse.

Graph	RLF	Saturation	Hybrid (32 Procs)	S-Impasse (32 Procs)
School-as.dat [324 nodes]	30 (4.88 s)	28 (14.96 s)	23 (6.62 m) [total: 1 hr]	23 (28.38 m) [total: 1 hr]
Model-1.dat [299 nodes]	32 (4.19 s)	31 (14.71 s)	28 (2.82 m) [total: 1 hr]	28 (4 m) [total: 1 hr]
$G_{299,0.23}$	22 (3.84 s)	22 (12.15 s)	16 (1.2 hrs) [total: 2 hr]	17 (1.14 hrs) [total: 1.2 hr]

Table 8: Register Allocation Graphs. In the leftmost column, graphs are described by file name (as in Challenge files), with chromatic number in parentheses. For each of RLF, Saturation and Hybrid, the following data is listed: (i) The size of the best coloring found by each algorithm. (ii) The number of runs achieving this coloring, over the total number of runs. (iii) The average running time, over all runs obtaining the best coloring.

Graph	RLF			Saturation			Hybrid(32 Procs)		
	(i)	(ii)	(iii)	(i)	(ii)	(iii)	(i)	(ii)	(iii)
mulsol.1 [197 nodes] (49)	49	(1/1)	(26.66 secs)	49	(1/1)	(34.00 secs)	49	(1/1)	(26.66 secs)
mulsol.2 [188 nodes] (31)	31	(1/1)	(26.48 secs)	31	(1/1)	(36.00 secs)	31	(1/1)	(26.48 secs)
mulsol.3 [184 nodes] (31)	31	(1/1)	(27.23 secs)	31	(1/1)	(32.92 secs)	31	(1/1)	(26.23 secs)
mulsol.4 [185 nodes] (31)	31	(1/1)	(26.5 secs)	31	(1/1)	(34.03 secs)	31	(1/1)	(26.5 secs)
mulsol.5 [186 nodes] (31)	31	(1/1)	(26.51 secs)	31	(1/1)	(33.52 secs)	31	(1/1)	(26.51 secs)
zeroin.1 [211 nodes] (49)	49	(1/1)	(26.95 secs)	49	(1/1)	(32.39 secs)	49	(1/1)	(26.95 secs)
zeroin.2 [211 nodes] (30)	30	(1/1)	(26.31 secs)	30	(1/1)	(36.40 secs)	30	(1/1)	(26.31 secs)
zeroin.3 [206 nodes] (30)	30	(1/1)	(26.32 secs)	30	(1/1)	(36.40 secs)	30	(1/1)	(26.32 secs)
fpsol2.1 [496 nodes] (65)	65	(1/1)	(28.99 secs)	65	(1/1)	(28.95 secs)	65	(1/1)	(28.99 secs)
fpsol2.2 [451 nodes] (30)	30	(1/1)	(26.36 secs)	30	(1/1)	(25.49 secs)	30	(1/1)	(25.49 secs)
fpsol2.3 [425 nodes] (30)	30	(1/1)	(26.05 secs)	30	(1/1)	(25.6 secs)	30	(1/1)	(25.6 secs)
inithx.1 [864 nodes] (54)	54	(1/1)	(44.64 secs)	54	(1/1)	(42.32 secs)	54	(1/1)	(43.32 secs)
inithx.2 [645 nodes] (31)	31	(1/1)	(41.00 secs)	31	(1/1)	(39.20 secs)	31	(1/1)	(39.20 secs)
inithx.3 [621 nodes] (31)	31	(1/1)	(41.08 secs)	31	(1/1)	(39.00 secs)	31	(1/1)	(39.00 secs)

Table 9: Sparse Matrix Graphs. In the leftmost column, graphs are described by file name. When the optimal coloring size is known, this is noted in parentheses in the first column. For each of RLF, Saturation and Hybrid, the following data is listed: (i) The size of the best coloring found by each algorithm. (ii) The number of runs achieving this coloring, over the total number of runs. (iii) The average running time, over all runs obtaining the best coloring. These runs were done on a single CM-5 processor which requires much less time to do i/o than a parallel run. These tests were not done on the Hybrid, due to lack of memory (the parallel memory allocator greatly overallocates memory on the CM-5).

Graph	RLF			Saturation		
	(i)	(ii)	(iii)	(i)	(ii)	(iii)
s1084.dat(3)	4	(5/5)	(40 secs)	4	(5/5)	(6 secs)
s1993.dat(4)	3	(5/5)	(7 secs)	4	(5/5)	(9 secs)
s707.dat(10)	10	(5/5)	(9 secs)	10	(4/5)	(3 secs)
s147.dat(2)	2	(5/5)	(1 sec)	2	(5/5)	(1 sec)

Table 10: Exam Scheduling Graphs. In the leftmost column, graphs are described by file name. When the optimal coloring size is known, this is noted in parentheses in the first column. For each of RLF, Saturation and Hybrid, the following data is listed: (i) The size of the best coloring found by each algorithm. (ii) The number of runs achieving this coloring, over the total number of runs. (iii) The average running time, over all runs obtaining the best coloring.

Graph	RLF			Saturation			Hybrid(32 Procs)		
	(i)	(ii)	(iii)	(i)	(ii)	(iii)	(i)	(ii)	(iii)
fl-tech.1(11)	11	(5/5)	(23.4 secs)	11	(5/5)	(38.5 secs)	11	(5/5)	(23 secs)
fl-tech.2(8)	8	(5/5)	(23.5 secs)	8	(5/5)	(44 secs)	8	(5/5)	(23.5 secs)
fl-tech.3	6	(5/5)	(23.1 secs)	6	(5/5)	(37.5 secs)	6	(5/5)	(23.1 secs)

Table 11: Latin Square Graph. For each of RLF, Saturation and Hybrid, the following data is listed: (i) The size of the best coloring found by each algorithm. (ii) The number of runs achieving this coloring, over the total number of runs. (iii) The average running time, over all runs obtaining the best coloring.

Graph	RLF			Saturation			Hybrid(32 Procs)		
	(i)	(ii)	(iii)	(i)	(ii)	(iii)	(i)	(ii)	(iii)
Latin Square	146	(2/4)	(9.8 mins)	132	(5/5)	(2.5 mins)	109	(3/4)	(1.9 hrs)

Table 12: Comparison of number of iterations needed by parallel and sequential S-Impasse algorithms to decrease colorings, for three graphs.

Graph	Coloring Size	Parallel S-Impasse		Sequential S-Impasse	
$G_{70,0.5}$	14	0	(0.35 secs)	0	(0.24 secs)
	13	12	(7.82 secs)	21	(1.31 secs)
	12	92	(13.69 secs)	557	(5.96 secs)
	11	523	(29.13 secs)	7757	(98.27 secs)
le-450.15c	29	1	(start)	434	(37.1 secs)
	27	350	(55.5 secs)	2770	(196.3 secs)
	25	4915	(263.9)	11963	(651.9 secs)
	23	17663	(836.0 secs)	51260	(3086.3 secs)
	22	41861	(1706.2 secs)		
	21	148349	(4590.1 secs)		
school-as.dat	28	48	(19.98 s)	0	(3.58 s)
	27	154	(20.73 s)	305	(10.76 s)
	26	823	(42.51 s)	686	(15.51 s)
	25	1789	(48.89 s)	4767	(66.43 s)

Table 13: Parallel Exhaustive-Search. The running time on a $G_{70,0.5}$ graph is given for various numbers of processors. Also presented for each number of processors are the total number of nodes expanded in the search tree, plus the minimum and maximum number expanded by a single processor.

No. Procs.	Running Time	Total Nodes Expanded	Min Nodes Expanded	Max Nodes Expanded
1	19.5 mins	323,881 (100%)	323,881	323,881
2	10.7 mins	334,156 (103.2%)	163,040	171,116
4	6.08 mins	327,621 (101.2%)	73,556	99,279
8	5.48 mins	359,681 (111.1%	35,234	79,744
16	5.22 mins	340,893 (105.3%)	12,382	73,925

Appendix II

Second DIMACS Challenge

Coloring Benchmark Results

GENERAL INFORMATION Authors: Gary Lewandowski and Anne Condon

Title: Experiments with Parallel Graph Coloring Heuristics

Name of Algorithm: Hybrid

Brief Description of Algorithm: Heuristic: Parallel Hybrid of parallel branch and bound exhaustive search algorithm and parallel S-Impasse algorithm.

Type of Machine: Connection Machine CM-5
Compiler and flags used: g++, -g flag

MACHINE BENCHMARKS

User time for instances:

r100.5	r200.5	r300.5	r400.5	r500.5
1.83	14.38	122.88	773.39	2993.58

ALGORITHM BENCHMARKS

Authors' Comments: Each run was time bounded by three hours. We consider machine crashes before three hours to be failures, with the exception of R1000.5.col which always crashed after 15 minutes (for unknown reasons) so the results are the best found in that period of time. The C2000 and C4000 graphs did not run due to lack of memory on the CM-5. (This is partially a problem of size and the memory allocator which allocates too much memory at a time.)

Results on Benchmark Instances

		Time			Solution		
Name	Runs (Fail)	Min	Avg (Std. Dev.)	Max	Min	Avg (Std. Dev.)	Max
DSJC125.5.col	5	1199.22	4043.63(2508.8)	8037.7	17	17(0)	17
DSJC250.5.col	5	306.106	4358.12(2789.37)	8013.02	29	29.2(0.447214)	30
DSJC500.5.col	5	1172.94	4783.86(2715.41)	7866.71	52	53(0.707107)	54
DSJC1000.5.col	5	4171.87	5333.81(1644.59)	8232	99	100(0.707107)	101
C2000.5.col	5 (5)						
C4000.5.col	5 (5)						
R125.1.col	5	50	64.6(22.2666)	104	5	5(0)	5
R125.1c.col	5	60	85(22.6716)	120	46	46(0)	46
R125.5.col	5	31.47	32.986(1.98744)	36.38	37	37(0)	37
R250.1.col	5	22	22(0)	22	8	8(0)	8
R250.1c.col	5	110.7	278.16(160.948)	505.8	64	64(0)	64
R250.5.col	5	38.9	39.88(0.563028)	40.3	66	66(0)	66
DSJR500.1.col	5	24.5	26.64(4.11983)	34	12	12(0)	12
DSJR500.1c.col	5 (1)	1331.1	5767.67(3703.33)	10139.6	85	85.25(0.5)	86
DSJR500.5.col	5	85.6	90.5(5.245)	96.2	128	128(0)	128
R1000.1.col	5	49.4	49.88(0.268328)	50	20	20(0)	20
R1000.1c.col	5	259.4	3940(5009.89)	10178	101	102.6(1.14018)	104
R1000.5.col	5	210	215.9(4.9548)	223.5	243	245.6(1.51658)	247
flat300_20_0.col	5	236.8	274.3(36.6314)	329.3	20	20(0)	20
flat300_26_0.col	5	2721.3	6637.14(3654.59)	10518.1	32	32.4(0.547723)	33
flat300_28_0.col	5	454.6	1913.54(1515.12)	3786.9	33	33(0)	33
flat1000_50_0.col	5	7172.8	7792.66(503.902)	8374.7	96	97(0.707107)	98
flat1000_60_0.col	5	3766	6288.36(1469.67)	7518.7	97	97.8(0.83666)	99
flat1000_76_0.col	5 (1)	5697	6497.85(664.774)	7100.3	99	99(0)	99
latin_square_10.col	5 (1)	5266.4	6520.12(1532.44)	8588.5	109	109.25(0.5)	110
le450_15a.col	5	88	162.62(75.522)	278.1	15	15(0)	15
le450_15b.col	5	113.3	178.36(45.0573)	226.1	15	15(0)	15
le450_15c.col	5	1016.1	2229.61(1114.42)	3828.8	16	16.6(0.547723)	17
le450_15d.col	5	1303.5	2859.6(1999.92)	5754.8	16	16.8(0.447214)	17
mulsol.i.1.col	5	27	27.2(0.447214)	28	49	49(0)	49
school1.col	5	37.6	46.26(7.82292)	55.2	14	14(0)	14
school1_nsh.col	5	54.2	66.4(9.36616)	76.4	14	14(0)	14