Parallel Graph Coloring Algorithms Using OpenMP

Extended Abstract

Assefaw Hadish Gebremedhin* Fredrik Manne†

1 Introduction

The graph coloring problem (GCP) deals with assigning labels (called colors) to the vertices of a graph such that adjacent vertices do not get the same color. The primary objective is to minimize the number of colors used. The GCP arises in a number of scientific computing and engineering applications. Examples include time tabling and scheduling [11], frequency assignment [6], register allocation [3], printed circuit testing [8], parallel numerical computation [1], and optimization [4]. Coloring a general graph with the minimum number of colors is known to be an NP-hard problem [7], thus one often relies on heuristics to compute a solution.

In a parallel application a graph coloring is usually performed in order to partition the work associated with the vertices into independent subtasks such that the subtasks can be performed concurrently. Depending on the amount of work associated with each vertex there are basically two coloring strategies one can use. The first strategy emphasizes on *minimizing* the number of colors and the second on *speed*. As to which is more appropriate depends on the underlying problem one is trying to solve.

If the task associated with each vertex is computationally expensive then it is crucial to use as few colors as possible. There exist several computation intensive local improvement heuristics for addressing this need. Some of these heuristics are also highly parallelizable [11].

If on the other hand, the task associated with each vertex is fairly small and one repeatedly has to find new graph colorings then the overall time to perform the colorings might take up a significant portion of the entire computation. See [13] for an example of this case. In such a setting it is important to compute a coloring fast and minimizing the number of colors used becomes less important. For this purpose there exist several linear time, or close to linear time, sequential greedy coloring heuristics. These heuristics have been found to be effective in coloring graphs that arise from a number of applications [4, 10].

This paper deals mainly with the latter problem of developing fast sublinear parallel coloring algorithms. Previous work on developing such algorithms has been performed on distributed memory computers using explicit messsage-passing. The speedup obtained so far has been discouraging [1]. The main justification for using these algoritms has been access to more memory and thus the potential to store large graphs. We note that the current availability of shared memory computers where the entire memory can be accessed by any processor makes this argument less significant now.

With the development of shared memory computers have also followed new programming paradigms of which OpenMP has become one of the most succesfull and widely used [15].

In this paper we present a fast and scalable parallel graph coloring algorithm suitable for the shared memory programming model. Our algorithm is based on first performing a parallel pseudo-coloring of the graph. This coloring might contain adjacent vertices that are colored with the same color. To remedy this we perform a second parallel step where any inconsistencies in the coloring are detected. These are then resolved in a final sequential step. An analysis on the PRAM model shows

*Both authors: Department of Informatics, University of Bergen, N-5020 Bergen, Norway, email:assefaw@ii.uib.no

†Fredrik.Manne@ii.uib.no

that the expected number of conflicts from the first stage is low and for $p = o(\frac{n}{\sqrt{m}})$ the algorithm is expected to provide a nearly linear speedup, where p is the number of processors used and n and m are the number of vertices and edges respectively. We also extend this idea and present a parallel algorithm that improves on a given coloring.

The presented algorithms have been implemented using OpenMP on a Cray Origin 2000. Experimental results on a number of very large graphs show that the algorithms yield good speedup and produce colorings of comparable quality to that of their sequential counterparts. The fact that we are using OpenMP to parallelize our program makes our implementation much simpler and easier to verify than if we had used a distributed memory programming environment such as MPI.

The rest of this paper is organized as follows. In Section 2 we give some background on the graph coloring problem and previous efforts to parallelize it. In Section 3 we describe our new parallel graph coloring algorithms and analyze their performance on the PRAM model. In Section 4 we present and discuss results from experiments performed on the Cray Origin 2000. Finally, in Section 5 we give concluding remarks.

2 Background

In this section we give a brief overview of previous work on developing fast sequential and parallel coloring algorithms. We also introduce some graph notations used in this paper.

For a graph $G = (V, E)$, we denote $|V|$ by n, $|E|$ by m, and the degree of vertex v_i by $deg(v_i)$. Moreover, the maximum, minimum, and average degree in a graph G are denoted by Δ, δ, and $\overline{\delta}$ respectively.

As mentioned in Section 1 there exist several fast sequential coloring heuristics that are very effective in practice. These algorithms are all based on one general greedy framework: A vertex is selected according to some predefined criterion and then colored with the smallest valid color. The selection and coloring continues until all the vertices in the graph are colored.

Some of the suggested coloring heuristics under this general framework include Largest degree-First-Ordering (LFO) [16], Saturation-Degree-Ordering (SDO) [2], and Incidence-Degree-Ordering (IDO) [4]. These heuristics choose at each step a vertex v with the maximum "degree" of some form among the set of uncolored vertices. In LFO, the standard definition of degree of a vertex is used. In IDO, incidence degree is defined as the number of already colored adjacent vertices whereas in SDO one only considers the number of differently colored adjacent vertices. First Fit (FF) is yet another, simple variant of the general greedy framework. In FF, the next vertex from some arbitrary ordering is chosen and colored. Intuitively, in terms of quality of coloring, these heuristics can roughly be ranked in an increasing order as FF, LFO, IDO, and SDO. Note that for a graph G the number of colors used by any sequential greedy algorithm is bounded from above by $\Delta + 1$. On the average, it has been shown that for *random* graphs FF is expected to use no more than $2\chi(G)$ colors, where $\chi(G)$ is the chromatic number of G [9]. In terms of run time, FF is clearly $O(m)$, LFO and IDO can be implemented to run in $O(m)$ time, and SDO in $O(n^2)$ time [10, 2].

When it comes to parallel graph coloring, a number of the existing fast heuristics are based on the observation that an independent set of vertices can be colored in parallel. Algorithm 1 outlines a general parallel heuristic based on this observation.

Depending on how the independent set is chosen and colored, Algorithm 1 specializes into a number of variants. The Parallel Maximal Independent set (PMIS) coloring is one variant. This is a heuristic based on Luby's maximal independent set finding algorithm [12]. Other variants

Algorithm 1

$ParallelColoring(G = (V, E))$
begin
  $U \leftarrow V$
  $G' \leftarrow G$
  while (G' is not empty) do in parallel
    Find an independent set I in G'
    Color the vertices in I
    $U \leftarrow U \setminus I$
    $G' \leftarrow$ graph induced by U
  end-while
end

Figure 1: A parallel coloring heuristic

are the asynchronous parallel heuristic by Jones and Plassmann (JP) [10], and the Largest-Degree-First(LDF) heuristic by Allwright et al. [1].

All of these algorithms are developed for distributed memory parallel computers. Allwright et al. made an experimental, comparative study by implementing the PMIS, JP, and LDF coloring algorithms on both SIMD and MIMD parallel architectures [1]. They report that they did not get any speedup for any of the algorithms.

Jones and Plassmann [10] do not report on obtaining speedup for their algorithms either. They state that "the running time of the heuristic is only a slowly increasing function of the number of processors used".

3 New Parallel Graph Coloring Heuristics

In this section we present two new parallel graph coloring heuristics and analyze their performance on the PRAM model. Our heuristics are based on block partitioning – dividing the vertex set (given in an arbitrary order) into p successive blocks of equal size. No effort is made to minimize the number of *crossing* edges i.e., edges whose end points belong to different blocks. Obviously, because of the existence of crossing edges, the coloring subproblems defined by each block are not independent.

3.1 A New Parallel Algorithm

The strategy we employ consists of three phases. In the first phase, the input vertex set V of graph $G = (V, E)$ is partitioned into p blocks as $\{V_1, V_2, \ldots, V_p\}$ such that $\lfloor \frac{n}{p} \rfloor \leq |V_i| \leq \lceil \frac{n}{p} \rceil$, $1 \leq i \leq p$. The vertices in each block are then colored in parallel using p processors. When coloring a vertex, *all* its previously colored neighbors, both the local ones and those found on other blocks, are taken into account. In the concurrent coloring, two processors may simultaneously be attempting to color vertices that are adjacent to each other. If these vertices are given the same color, the resulting coloring becomes invalid and hence we call the coloring obtained a *pseudo coloring*. In the second phase, each processor p_i checks whether vertices in V_i are assigned valid colors by comparing the color of a vertex against all its neighbors, both local and non-local. This checking step is also done in parallel. If a conflict is discovered, one of the endpoints of the edge in conflict is stored in a table. Finally, in the third phase, the vertices stored in this table are colored sequentially. Algorithm 2 provides the details of this strategy.

Algorithm 2

BlockPartitionBasedColoring(G, p)
begin
1. Partition V into p equal blocks $V_1 \ldots V_p$,
   where $\lfloor \frac{n}{p} \rfloor \leq |V_i| \leq \lceil \frac{n}{p} \rceil$
   for $i = 1$ to p do in parallel
     for each $v_j \in V_i$ do
       assign the smallest legal color
       to vertex v_j
     end-for
   end-for
2. for $i = 1$ to p do in parallel
     for each $v_j \in V_i$ do
       for each neighbor u of v_j do
         if $color(v_j) = color(u)$ then
           store min $\{u, v_j\}$ in the array A
         end-if
       end-for
     end-for
   end-for
3. Color the vertices in A sequentially
end

Figure 2: Block partition based coloring

3.1.1 Analysis

Our anaysis is based on the PRAM model where we assume that processors involved in the parallel computation operate in locksteps. In Algorithm 2, this amounts to saying that at each time unit t_j, processor p_i colors vertex $v_j \in V_i$, $1 \leq j \leq \lceil n/p \rceil$.

Our first result gives an upper bound on the *expected* number of conflicts (denoted by K) obtained at the end of Phase 2 of Algorithm 2.

Lemma 3.1 *The expected number of conflicts at the end of Phase 2 of Algorithm 2 is at most* $o(\overline{\delta}p)$

Proof: Consider a vertex $x \in V$ that is colored at time unit t_j, $1 \leq j \leq n/p$. Assuming that

the neighbors of x are randomly distributed, the *expected* number of neighbors of x that are concurrently colored at time unit t_j is given by

$$\frac{p-1}{n-1}deg(x) \tag{1}$$

If we sum (1) over all vertices in G we count each *potential* conflict twice. The expected number of conflicts is therefore bounded as follows.

$$\begin{aligned} E[K] &\leq (1/2)\sum_{x\in V}\frac{p-1}{n-1}deg(x) && (2)\\ &= (1/2)\frac{p-1}{n-1}(2m) && (3)\\ &= (1/2)\overline{\delta}(p-1)(n/n-1) && (4)\\ &= o(\overline{\delta}p) && (5)\end{aligned}$$

In going from (3) to (4), the identity $\overline{\delta} = \frac{\sum_{v\in V} deg(v)}{n} = \frac{2m}{n}$ is used.

□

We note that the result from Lemma 3.1 is pessimistic. For two adjacent vertices x and y colored at time t_j to get the same color c_i they must both have already colored neighbors with colors c_1 through c_{i-1} but not c_i.

We now look at the expected[1] run time. To do so, we introduce a graph attribute called *relative sparsity* r, defined as $r = \frac{n^2}{m}$. The attribute r indicates how sparse the graph is, the higher the value of r, the sparser the graph is. The following Lemma states that for most sparse graphs and realistic choices of p, Algorithm 2 provides an almost linear speedup compared to the sequential First Fit algorithm.

Lemma 3.2 *On a CREW PRAM, Algorithm 2 colors the input graph consistently in $EO(\Delta n/p)$ time when $p = O(\sqrt{r})$ and in $EO(\Delta\overline{\delta}p)$ time when $p = \omega(\sqrt{r})$.*

Proof: Note first that since Phase 3 resolves all the conflicts that are inherited from Phase 2, the coloring at the end of Phase 3 is a valid one. Both Phase 1 and 2 require concurrent read capability and thus the required PRAM is CREW. We then look at the run time. The overall time required by Algorithm 2 is $T = T_1+T_2+T_3$, where T_i is the time required by Phase i. Both Phase 1 and 2 consist of n/p parallel steps. The number of operations in each parallel step is proportional to the degree of the vertex under investigation. The degree of each vertex is bounded from above by Δ. Thus, $T_1 = T_2 = EO(\Delta n/p)$. The time required by the sequential step (Phase 3) is $T_3 = EO(\Delta K)$ where K is the number of conflicts at the end of Phase 2. From Lemma 3.1, $E[K] = o(\overline{\delta}p)$. Substituting yields,

$$T = T_1 + T_2 + T_3 = EO(\Delta n/p + \Delta\overline{\delta}p) \tag{6}$$

We now investigate two cases depending on the value of p.

Case I: $p = O(\sqrt{r})$

Using the definition $r = \frac{n^2}{m}$ this case can be restated as

$$p^2 \leq c\frac{n^2}{m} \tag{7}$$

where c is a constant. Multiplying both side of (7) by $2m/np$ we get

$$2mp/n \leq 2cn/p \tag{8}$$

Using the identity $\overline{\delta} = \frac{2m}{n}$, (8) can be written as

$$\overline{\delta}p = O(n/p) \tag{9}$$

In this case the first term in (6) dominates, and thus $T = EO(\Delta n/p)$ as claimed.

Case II: $p = \omega(\sqrt{r})$

Similar steps as in Case I can be used to reduce this condition to

$$\overline{\delta}p = \omega(n/p) \tag{10}$$

In this case the second term in (6) dominates, and thus $T = EO(\Delta\overline{\delta}p)$ as claimed. This completes our proof.

□

3.2 Reducing the Number of Colors

In this section we show how Algorithm 2 can be modified to use fewer colors. This is motivated by the idea behind Culberson's Iterated Greedy (IG) coloring heuristic [5]. IG is based on the following result, stated here without proof.

[1] Expected time complexity expressions are identified by the prefix E.

Lemma (Culberson) 3.3 *Let C be a k-coloring of a graph G, and π a permutation of the vertices such that if $C(v_{\pi(i)}) = C(v_{\pi(m)}) = c$, then $C(v_{\pi(j)}) = c$ for $i < j < m$. Then, applying the First Fit algorithm to G where the vertices have been ordered by π will produce a coloring using k or fewer colors.*

From Lemma 3.3, we see that if FF is reapplied on a graph where the vertex set is ordered such that vertices belonging to the same color class[2] in a previous coloring are listed consecutively, the new coloring is better or at least as good as the previous coloring. There are many ways in which the vertices of a graph can be arranged satisfying the condition of Lemma 3.3. One such ordering is the reverse color class ordering [5]. In this ordering, the color classes are listed in reverse order of their introduction. This has a potential for producing an improved coloring since the new one proceeds by first coloring vertices that could not be colored with low values previously.

The improved coloring heuristic has one more phase than Algorithm 2. The first phase is the same as Phase 1 of Algorithm 2. Let the coloring number used by this phase be $ColNum$. During the second phase, the pseudo coloring of the first phase is used to get a reverse color class ordering of the vertices. The second phase consists of $ColNum$ steps. In each step i, the vertices of color class $ColNum - i - 1$ are colored *afresh* in parallel. The remaining two phases are the same as Phases 2 and 3 of Algorithm 2. The method just described is outlined in Algorithm 3.

Each color class at the end of Phase 1 is a pseudo independent set. Hence block partitioning of the vertices of each color class results in only a few crossing edges. In other words, the number of conflicts expected at the end of Phase 2 (K_2) should be smaller than the number of conflicts at the end of Phase 1 (K_1). Thus, in addition to improving the quality of coloring, Phase 2 should also provide a reduction in the number of conflicts. Note that a conflict removing step is included in Phase 4 to ensure that any remaining conflicts are removed.

The following result shows that Phase 2 reduces the upper bound on the number of conflicts from Phase 1 by a factor of $\Theta(\Delta p/n)$.

[2] Vertices of the same color constitute a color class

Algorithm 3

ImprovedBlockPartitionBasedColoring(G, p)
begin
1. As Phase 1 of Algorithm 2
   {At this point we have the pseudo independent sets ColorClass(1) ... ColorClass(ColNum) }
2. for $k = ColNum$ down to 1 do
   Partition $ColorClass(k)$ into p equal blocks $V'_1 \ldots V'_p$
   for $i = 1$ to p do in parallel
   for $j = 1$ to $|ColorClass(k)|/p$ do
   assign the smallest legal color to vertex $v_j \in V'_i$
   end-for
   end-for
   end-for
3. As Phase 2 of Algorithm 2
4. As Phase 3 of Algorithm 2

end

Figure 3: Modified block partition based coloring

Lemma 3.4 *The upper bound on the expected number of conflicts at the end of Phase 2 of Algorithm 3 is reduced by a factor of $\Theta(\Delta p/n)$ compared with the upper bound on the number of conflicts from Phase 1.*

Proof: The proof is similar to that of Lemma 3.1 and omitted here to save space.

4 Experimental Results

In this section, we experimentally demonstrate the performance of the algorithms developed in Section 3. The experiments have been performed on the shared memory parallel computer Cray Origin 2000. The new algorithms have been implemented in Fortran 90 and parallelized using OpenMP[15]. We have also implemented the sequential versions of FF and IDO to use as benchmarks for comparing the number of colors used by our parallel coloring heuristics.

Synchronous mode of operation was assumed in the analysis in Section 3. In our implementation, however, no synchronization was made.

The test graphs used in our experiments arise from practical applications. They are divided into three categories as Problem Set I, II, and III (corresponding to the partitioning in Table 1). Problem Sets I and II consist of graphs (matrices) that arise from finite element methods [14]. Problem Set III consists of matrices that arise in eigenvalue computations [13].

Table 1 provides some statistics about the test graphs and the number of colors required to color them using sequential FF and IDO (shown under columns χ_{FF} and χ_{IDO} respectively).

Table 2 lists results obtained using our first parallel coloring heuristic(Algorithm 2). The number of blocks (processors) used is given in column p. Columns χ_1 and χ_3 give the number of colors used at the end of Phases 1 and 3 respectively. The number of conflicts that arise in Phase 1 are listed under the column labeled K. The column labeled $\overline{\delta}(p-1)$ indicates the theoretically expected upper bound on the number of conflicts as predicted by Lemma 3.1. The time in milliseconds required by the different phases are listed under T_1, T_2, T_3, and the last column T_{tot} gives the total time used. The column labeled S_{par} lists the speedup obtained compared to the time used by running Algorithm 2 on 1 processor ($S_{par}(p) = \frac{T_{tot}(1)}{T_{tot}(p)}$). The last column, S_{seqFF}, gives the speedup obtained by comparing against a straight forward sequential FF ($S_{seqFF}(p) = \frac{T_1(1)}{T_{tot}(p)}$).

The results in column K of Table 2 show that in general, the number of conflicts that arise in Phase 1 is small and grows as a function of the number of blocks (or processors) p. This agrees well with the result from Lemma 3.1. We see that for the relatively dense graphs the actual number of conflicts is much less than the bound given by Lemma 3.1.

The run times obtained show that Algorithm 2 performs as predicted by Lemma 3.2. Particularly, the time required for recoloring incorrectly colored vertices is observed to be practically zero for all our test graphs. This is not surprising as the obtained value of K is negligibly small compared to the number of vertices in a given graph.

As results in columns T_1 and T_2 indicate, the time used to detect conflicts is approximately the same as the time used to do the initial coloring. This makes the running time of the algorithm using one processor approximately double that of the sequential FF. This in turn reduces the speedup obtained compared to the sequential FF by a factor of 2. The speedup obtained compared to the parallel algorithm using one processor obtains its best values for the two largest graphs mrng3 and dense2.

Table 3 lists results of Algorithm 3. The number of colors used at the end of Phases 1 and 2 are listed in columns χ_1 and χ_2, respectively. The coloring at the end of Phase 2 is not guaranteed to be conflict-free. Phases 3 and 4 detect and resolve any remaining conflicts. Column χ_4 lists the number of colors used at the end of Phase 4. The number of conflicts at the end of Phases 1 and 2 are listed under K_1 and K_2, respectively. The time elapsed (in milliseconds) at the various stages are given in columns T_1, T_2, T_3, T_4, and T_{tot}. Speedup values in column S_{par} are calculated as in the corresponding column of Table 2. The column S_{2seqFF} gives speedups as compared to a two-run of sequential FF ($S_{2seqFF} = \frac{T_1(1)+T_2(1)}{T_{tot}(p)}$).

Results in column χ_2 confirm that Phase 2 of Algorithm 3 reduces the number of colors used by Phase 1. This is especially true for test graphs from Problem Sets II and III, which contain relatively denser graphs than Problem Set I. It is interesting to compare the results in column χ_2 with the results in the χ_{IDO} column of Table 1. We see that in general the quality of the coloring obtained using Algorithm 3 is comparable with that of the IDO algorithm. IDO is known to be one of the most effective coloring heuristics [4].

From column K_2 we see that the number of conflicts that remain after Phase 2 of Algorithm 3 is zero for almost all test graphs and values of p. The only occasion where we obtained a value other than zero for K_2 was using $p = 12$ for the graphs dense1 and dense2. These results agree well with the claim in Lemma 3.4.

5 Conclusion

We have presented a new parallel coloring heuristic suitable for shared memory programming. The heuristic is fast and simple and yields good speedup for graphs of practical interest and on a realistic number of processors. We have also introduced a second heuristic that can improve on the quality of coloring obtained from the first one. Experimental

results conducted on both heuristics using OpenMP validate the theoretical analysis performed using the PRAM model.

One of the main arguments against using OpenMP to parallelize code has been that it does not give as good speedup as a more dedicated message passing implementation using MPI. The results in this paper show an example where the opposite is true, the OpenMP algorithms have better speedup than existing message passing based algorithms. Moreover, implementing the presented algorithms in a message passing environment would have required a considerable effort and it is not clear if this would have led to efficient algorithms. It has been a relativly straight forward task to implement these algorithms using OpenMP as all the communication is hidden from the programmer.

We believe that the method used in these coloring heuristics can be applied to develop parallel algorithms for other graph problems and we are currently investigating this in problems related to sparse matrix computations.

References

[1] J. R. Allwright, R. Bordawekar, P. D. Coddington, K. Dincer, and C. L. Martin. A comparison of parallel graph coloring algorithms. Technical Report Tech. Rep. SCCS-666, Northeast Parallel Architecture Center, Syracuse University, 1995.

[2] D. Brelaz. New methods to color the vertices of a graph. *Comm. ACM*, 22(4), 1979.

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

[4] T.F. Coleman and J.J. More. Estimation of sparse jacobian matrices and graph coloring problems. *SIAM Journal on Numerical Analysis*, 20(1):187–209, 1983.

[5] Joseph C. Culberson. Iterated greedy graph coloring and the difficulty landscape. Technical Report TR 92-07, Department of Computing Science, The University of Alberta, Edmonton, Alberta, Canada, June 1992.

[6] Andreas Gamst. Some lower bounds for a class of frequency assignment problems. *IEEE transactions of Vehicular Technology*, 35(1):8–14, 1986.

[7] M.R. Garey and D.S. Johnson. *Computers and Intractability.* W.H. Freeman, New York, 1979.

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

[9] G.R. Grimmet and C.J.H. McDiarmid. On coloring random graphs. *Mathematical Proceedings of the Cambridge Philsophical Society*, 77:313–324, 1975.

[10] Mark T. Jones and Paul E. Plassmann. A parallel graph coloring heuristic. *SIAM journal of scientific computing*, 14(3):654–669, May 1993.

[11] Gary Lewandowski. *Practical Implementations and Applications Of Graph Coloring.* PhD thesis, University of Wisconsin-Madison, August 1994.

[12] M. Luby. A simple parallel algorithm for the maximal independent set problem. *SIAM journal on computing*, 15(4):1036–1053, 1986.

[13] Fredrik Manne. A parallel algorithm for computing the extremal eigenvalues of very large sparse matrices (extended abstract). In *proceedings of Para98*, volume 1541, pages 332–336. Lecture Notes in Computer Science, Springer, 1998.

[14] na. *ftp://ftp.cs.umn.edu/users/kumar/Graphs/.*

[15] OpenMP. A proposed industry standard api for shared memory programming. *http://www.openmp.org/.*

[16] D.J.A. Welsh and M.B. Powell. An upper bound for the chromatic number of a graph and its application to timetabling problems. *Computer Journal*, (10):85–86, 1967.

Problem	n	m	Δ	δ	$\overline{\delta}$	$\sqrt{r}$	χ_{FF}	χ_{IDO}
mrng2	1,017,253	2,015,714	4	2	3	716	5	5
mrng3	4,039,160	8,016,848	4	2	3	1426	5	5
598a	110,971	741,934	26	5	13	128	11	9
m14b	214,765	1,679,018	40	4	15	165	13	10
dense1	19,703	3,048,477	504	116	309	11	122	122
dense2	218,849	121,118,458	1,640	332	1,106	20	377	376

Table 1: Test Graphs

Problem	p	χ_1	χ_3	K	$\overline{\delta}(p-1)$	T_1	T_2	T_3	T_{tot}	S_{par}	S_{seqFF}
mrng2	1	5	5	0	0	1190	1010	0	2200	1	0.6
mrng2	2	5	5	0	3	1130	970	0	2100	1.1	0.6
mrng2	4	5	5	0	9	430	280	0	710	3.1	1.7
mrng2	8	5	5	8	21	260	200	0	460	4.8	2.6
mrng2	12	5	5	18	33	200	130	0	330	6.7	3.6
mrng3	1	5	5	0	0	4400	3400	0	7800	1	0.6
mrng3	2	5	5	2	3	2250	1600	0	3850	2	1.1
mrng3	4	5	5	4	9	1300	1000	0	2300	3.4	1.9
mrng3	8	5	5	0	21	630	800	0	1430	5.5	3.1
mrng3	12	5	5	12	33	430	480	0	910	8.6	4.8
598a	1	11	11	0	0	100	80	0	180	1	0.6
598a	2	12	12	4	13	55	40	0	95	2	1.1
598a	4	12	12	12	39	40	20	0	60	3	1.7
598a	8	12	12	36	91	28	15	0	43	4.2	2.3
598a	12	12	12	42	143	20	15	0	35	5.2	2.9
m14b	1	13	13	0	0	200	180	0	380	1	0.5
m14b	2	13	13	2	15	130	120	0	250	1.5	0.8
m14b	4	14	14	14	45	80	50	0	130	3	1.5
m14b	8	13	13	16	105	48	26	0	74	5	2.7
m14b	12	13	13	36	165	40	20	0	60	6.4	3.3
dense1	1	122	122	0	0	200	290	0	490	1	0.4
dense1	2	142	142	30	309	110	140	0	250	2	0.8
dense1	4	137	137	94	927	69	72	0	141	3.5	1.4
dense1	8	129	129	94	2163	53	44	1	97	5.6	2.1
dense1	12	121	124	78	3399	55	90	1	145	3.4	1.4
dense2	1	377	377	0	0	9200	13200	0	22400	1	0.4
dense2	2	382	382	68	1106	5160	8040	3	13203	1.7	0.7
dense2	4	400	400	98	3318	2600	4080	4	6684	3.4	1.4
dense2	8	407	407	254	7742	1590	2280	11	3881	5.8	2.4
dense2	12	399	399	210	12166	1090	1420	8	2518	9	3.7

Table 2: Experimental results for Algorithm 2

Problem	p	χ_1	χ_2	χ_4	K_1	K_2	T_1	T_2	T_3	T_4	T_{tot}	S_{par}	S_{2seqFF}
mrng2	1	5	5	5	0	0	1050	1700	820	0	3570	1	0.8
mrng2	2	5	5	5	0	0	950	1350	650	0	2650	1.4	1.0
mrng2	4	5	5	5	2	0	470	840	310	0	1620	2.2	1.7
mrng2	8	5	5	5	16	0	300	500	200	0	1000	3.6	2.8
mrng2	12	5	5	5	12	0	250	400	170	0	820	4.4	3.4
mrng3	1	5	5	5	0	0	3700	9500	2600	0	15800	1	0.8
mrng3	2	5	5	5	0	0	1890	4100	1200	0	7190	2.2	1.8
mrng3	4	5	5	5	0	0	1100	2700	750	0	4550	3.5	2.9
mrng3	8	5	5	5	4	0	540	1800	450	0	2790	5.6	4.7
mrng3	12	5	5	5	24	0	450	1900	300	0	2650	6	5.0
598a	1	11	10	10	0	0	100	200	75	0	375	1	0.8
598a	2	12	10	10	14	0	65	105	37	0	207	1.8	1.5
598a	4	11	10	10	22	0	35	90	20	0	145	2.6	2.1
598a	8	12	11	11	40	0	30	99	25	0	154	2.4	2.0
598a	12	12	11	11	50	0	30	110	15	0	155	2.4	2.0
m14b	1	13	11	11	0	0	200	520	190	0	910	1	0.8
m14b	2	13	12	12	2	0	105	240	80	0	425	2.1	1.7
m14b	4	14	12	12	6	0	70	160	40	0	270	3.4	2.7
m14b	8	13	12	12	12	0	45	120	25	0	190	4.8	3.8
m14b	12	13	11	11	22	0	53	150	20	0	223	4	3.2
dense1	1	122	122	122	0	0	180	250	180	0	610	1	0.7
dense1	2	135	122	122	26	0	100	180	140	0	420	1.5	1.0
dense1	4	132	122	122	40	0	80	100	70	0	250	2.5	1.7
dense1	8	126	122	122	104	0	70	80	30	0	180	3.4	2.4
dense1	12	123	121	122	150	2	40	760	30	0	830	0.7	0.5
dense2	1	377	376	376	0	0	9920	13700	7500	0	31120	1	0.8
dense2	2	376	376	376	66	0	5200	6220	4200	0	15620	2	1.5
dense2	4	394	376	376	112	0	2700	3600	2100	0	8400	3.7	2.8
dense2	8	398	376	376	164	0	2000	2000	1800	0	5800	5.4	4.0
dense2	12	399	376	376	232	2	1100	1700	900	0	3700	8.4	6.4

Table 3: Experimental results for Algorithm 3