Approximating Interval Coloring and Max-Coloring in Chordal Graphs

SRIRAM V. PEMMARAJU, SRIRAM PENUMATCHA, and RAJIV RAMAN
University of Iowa

We consider two coloring problems: interval coloring and max-coloring for chordal graphs. Given a graph $G = (V, E)$ and positive-integral vertex weights $w : V \to \mathbf{N}$, the *interval-coloring* problem seeks to find an assignment of a real interval $I(u)$ to each vertex $u \in V$, such that two constraints are satisfied: (i) for every vertex $u \in V$, $|I(u)| = w(u)$ and (ii) for every pair of adjacent vertices u and v, $I(u) \cap I(v) = \emptyset$. The goal is to minimize the *span* $|\cup_{v \in V} I(v)|$. The *max-coloring problem* seeks to find a proper vertex coloring of G whose color classes $C_1, C_2, \ldots, C_k$, minimize the sum of the weights of the heaviest vertices in the color classes, that is, $\sum_{i=1}^{k} max_{v \in C_i} w(v)$. Both problems arise in efficient memory allocation for programs. The interval-coloring problem models the compile-time memory allocation problem and has a rich history dating back at least to the 1970s. The max-coloring problem arises in minimizing the total buffer size needed by a dedicated memory manager for programs. In another application, this problem models scheduling of conflicting jobs in batches to minimize the *makespan*. Both problems are NP-complete even for interval graphs, although there are constant-factor approximation algorithms for both problems on interval graphs. In this paper, we consider these problems for *chordal graphs*, a subclass of perfect graphs. These graphs naturally generalize interval graphs and can be defined as the class of graphs that have no induced cycle of length > 3. Recently, a 4-approximation algorithm (which we call `GeomFit`) has been presented for the max-coloring problem on perfect graphs (Pemmaraju and Raman 2005). This algorithm can be used to obtain an interval coloring as well, but without the constant-factor approximation guarantee. In fact, there is no known constant-factor approximation algorithm for the interval-coloring problem on perfect graphs. We study the performance of `GeomFit` and several simple $O(\log(n))$-factor approximation algorithms for both problems. We experimentally evaluate and compare four simple heuristics: first-fit, best-fit, `GeomFit`, and a heuristic based on partitioning the graph into vertex sets of similar weight. Both for max-coloring and for interval coloring, `GeomFit` deviates from OPT by about 1.5%, on average. The performance of first-fit comes close second, deviating from OPT by less than 6%, on average, for both problems. Best-fit comes third and graph-partitioning heuristic comes a distant last. Our basic data comes from about 10,000 runs of each of the heuristics for each of the two problems on randomly generated chordal graphs of various sizes, sparsity, and structure.

Categories and Subject Descriptors: D.4.2 [**Operating Systems**]: Storage Management; G.2.2 [**Graph Discrete Mathematics**]: Theory

Partially supported by National Science Foundation Grant DMS-0213305.
Authors' address: Sriram V. Pemmaraju, Sriram Penumatcha, and Rajiv Raman, Department of Computer Science, The University of Iowa, Iowa City, IA 52240-1419; email: [sriram,spenumat, rraman]@cs.uiowa.edu.

General Terms: Algorithms, Experimentation, Theory

Additional Key Words and Phrases: Graph coloring, chordal graphs, perfect graphs, dynamic storage allocation

1. INTRODUCTION

1.1 Interval Coloring

Given a graph $G = (V, E)$ and positive-integral vertex weights $w : V \to \mathbf{N}$, the *interval-coloring* problem seeks to find an assignment of an interval $I(u)$ to each vertex $u \in V$ such that two constraints are satisfied: (1) for every vertex $u \in V$, $|I(u)| = w(u)$ and (2) for every pair of adjacent vertices u and v, $I(u) \cap I(v) = \emptyset$. The goal is to minimize the span $|\cup_v I(v)|$. The interval-coloring problem has a fairly long history dating back, at least to the 1970s. For example, Stockmeyer showed in 1976 that the interval-coloring problem is NP-complete, even when restricted to interval graphs and vertex weights in $\{1, 2\}$ (see problem SR2 in Garey and Johnson [1979]). The main application of the interval-coloring problem is in the *compile-time memory-allocation problem*. Fabri [1979] made this connection in 1979. In order to reduce the total memory consumption of source-code objects (simple variables, arrays, and structures), the compiler can make use of the fact that the memory regions of two objects are allowed to overlap, provided that the objects do not "interfere" at run-time. This problem can be abstracted as the interval-coloring problem, as follows. The source-code objects correspond to vertices of our graph, run-time interference between pairs of source code objects is represented by edges of the graph, the amount of memory needed for each source-code object is represented by the weight of the corresponding vertex, and the assignment of memory regions to source code objects is represented by the assignment of intervals to vertices of the graph. Minimizing the size of the union of intervals corresponds to minimizing the amount of memory-allocation.

If we restrict our attention to straight-line programs, that is, programs without loops or conditional statements, then the compile-time memory-allocation problem can be modeled as the interval-coloring problem for interval graphs. This is because the interference graph for source-code objects in a straight-line program is an interval graph. As mentioned before, the interval-coloring problem is NP-complete for interval graphs and so research has focused on designing approximation algorithms for the problem. Kierstead [1988] gave the first constant-factor approximation algorithm—an 80-approximation, that was improved by Kierstead himself to a 6-approximation [Kierstead 1991]. This was followed by papers by Gergov [1996, 1999] who improved the approximation factor to 3. Recently, Buchsbaum et al. [2003] gave a $(2 + \varepsilon)$-approximation algorithm for the problem.

1.2 Max-Coloring

Like interval-coloring, the *max-coloring problem* takes as input a vertex-weighted graph $G = (V, E)$ with weight function $w : V \to \mathbf{N}$. The problem

requires that we find a proper vertex coloring of G whose color classes $C_1, C_2, \ldots, C_k$, minimize the sum of the weights of the heaviest vertices in the color classes, that is, $\sum_{i=1}^{k} max_{v \in C_i} w(v)$. The max-coloring problem models the problem of minimizing the total buffer size need for memory management in different applications. For example, [Govindarajan and Rengarajan 1996] uses max-coloring to minimize buffer size in digital signal-processing applications. In [Pemmaraju et al. 2004a], max-coloring models the problem of minimizing buffer size needed by memory managers for wireless protocol stacks like GPRS or 3G. In general, programs that run with stringent memory or timing constraints use a dedicated memory manager that provides better performance than the general-purpose memory management of the operating system. The most commonly used memory manager design for this purpose is the *segregated buffer pool*. This consists of a fixed set of buffers of various sizes with buffers of the same size linked together in a linked list. As each memory request arrives, it is satisfied by a buffer whose size is at least as large as the size of the memory request. The assignment of buffers to memory requests can be viewed as an assignment of colors to the requests—all requests that are assigned a buffer are colored identically. Requests that do not interfere with each other can be assigned the same color/buffer. Thus, the problem of minimizing the total size of the buffer pool corresponds to the max-coloring problem.

The max-coloring problem also arises in scheduling conflicting jobs in batches. Suppose we are given a set of jobs $\{J_1, \ldots, J_n\}$ with processing times $\{w_1, \ldots, w_n\}$ and a conflict relation between the jobs modeled as a graph with vertices representing the jobs and edges between vertices representing the conflict relation. The goal is to schedule this set of jobs in batches so that no two jobs in a batch can conflict and one batch can start only after the previous batch has been completed. Therefore, the processing time of each batch is the largest processing time of a job in that batch. This problem with the objective function of minimizing the *makespan* is the max-coloring problem. Other researchers have considered the batch-scheduling problem, but with the average completion time objective function [Bar-Noy et al. 2000].

Pemmaraju et al. [2004a] shows that the max-coloring problem is NP-complete for interval graphs and presents the first constant-factor approximation algorithm for the max-coloring problem for interval graphs. The paper makes a connection between max-coloring and on-line graph coloring and using a known result of Kierstead and Trotter [1981] on on-line coloring interval graphs, they obtain a 2-approximation algorithm for interval graphs and a 3-approximation algorithm for circular arc graphs. Recently, the first constant-factor approximation algorithm for perfect graphs (a 4-approximation algorithm) has appeared [Pemmaraju and Raman 2005]. This algorithm, which we call `GeomFit`, is one of the algorithms whose performance we experimentally evaluate in this paper.

1.3 Connections between Interval-Coloring and Max-Coloring

Given a coloring of a vertex-weighted graph $G = (V, E)$ with color classes $C_1, C_2, \ldots, C_k$, we can construct an assignment of intervals to vertices as follows.

For each i, $1 \leq i \leq k$, let $v_i \in C_i$ be the vertex with maximum weight in C_i. Let $H(1) = 0$, and for each $i, 2 \leq i \leq k$, let $H(i) = \sum_{j=1}^{i-1} w(v_j)$. For each vertex $v \in C_i$, we set $I(v) = (H(i), H(i)+w(v))$. Clearly, no two vertices in distinct color classes have overlapping intervals and, therefore, this is a valid interval-coloring of G. We say that this is the interval-coloring *induced* by the coloring $C_1, C_2, \ldots, C_k$. The span of this interval-coloring is $\sum_{i=1}^{k} w(v_i)$, which is the same as the weight of the coloring $C_1, C_2, \ldots, C_k$ viewed as a max-coloring. In other words, if there is a max-coloring of weight W for a vertex-weighted graph G, then there is an interval coloring of G of the same weight.

However, in Pemmaraju et al. [2004a], we show an instance of a vertex-weighted interval graph on n vertices for which the weight of an optimal max-coloring is $\Omega(\log n)$ times the weight of the heaviest clique. This translates into an $\Omega(\log n)$ gap between the weight of an optimal max-coloring and the span of an optimal interval-coloring because an optimal interval-coloring of an interval graph has span that is within $O(1)$ of the weight of a heaviest clique. In general, algorithms for max-coloring can be used for interval-coloring with minor modifications to make the interval assignment more "compact." While the worst-case performance of these algorithms can be bad for interval-coloring, our experiments show that, in practice, algorithms that do well for max-coloring do well for interval-coloring as well. For example, the performance of `GeomFit` for the interval-coloring provides strong evidence of this phenomenon. These connections motivate us to study interval-coloring and max-coloring in the same framework.

1.4 Chordal Graphs

For both the interval-coloring and max-coloring problems, the assumption that the underlying graph is an interval graph is somewhat restrictive. As mentioned before, in memory-allocation applications, if the underlying program is straight-line, then the corresponding conflicts can be modeled as an interval graph. However, most programs contain conditional statements and loops. In this paper we consider a natural generalization of interval graphs called chordal graphs. A graph is a *chordal graph* if it has no induced cycles of length 4 or more. Alternately, every cycle of length 4 or more in a chordal graph has a chord.

There are many alternate characterizations of chordal graphs. One that will be useful in this paper is the existence of a perfect elimination ordering of the vertices of any chordal graph. An ordering $v_n, v_{n-1}, \ldots, v_1$ of the vertex set of a graph is said to be a *perfect elimination ordering* if when vertices are deleted in this order, for each i, the neighbors of vertex v_i in the remaining graph, $G[\{v_1, v_2, \ldots, v_i\}]$ form a clique. A graph is a chordal graph iff it has a perfect elimination ordering. Tarjan and Yannakakis [1984] describe a simple linear-time algorithm called *maximum cardinality search* that can be used to determine if a given graph has a perfect elimination ordering and to construct such an ordering if it exists. Given a perfect elimination ordering of a graph G, the graph can be colored by considering vertices in reverse-perfect elimination order and assigning to each vertex the minimum available color. It is easy to

see that this greedy-coloring algorithm uses exactly as many colors as the size of the largest clique in the graph and, therefore, produces an optimal vertex coloring.

Every interval graph is also a chordal graph (but not vice versa). To see this, take an interval representation of an interval graph and order the intervals in left-to-right order of their left endpoints. It is easy to verify that this gives a perfect elimination ordering of the interval graph. Thus chordal graphs generalize interval graphs and one of our motivations in considering chordal graphs is to determine if the constant-factor algorithms for interval-coloring interval graphs can be extended to chordal graphs. Since the publication of the conference version of this paper, we have developed the first constant-factor approximation algorithm for max-coloring chordal graphs [Pemmaraju and Raman 2005]. Another motivation for considering chordal graphs is that the way certain kinds of compilers, such as algebraic compilers, process source code, the interference graph of source objects ends up being a chordal graph [Rus and Pemmaraju 1997]. A final motivation is that others have considered the problem of finding approximation algorithms for interval-coloring chordal graphs, but with limited success. For example, Confessore et al. [2002] shows a 2-approximation algorithm for the interval-coloring problem on claw-free chordal graphs, leaving the problem open for chordal graphs, in general.

1.5 Organization of this Paper

In this paper, we consider four simple heuristics for the interval-coloring and max-coloring problems and experimentally evaluate their performance.

These heuristics are:

- GeomFit. Vertices are colored in rounds. In each round, a subgraph of the graph is chosen and optimally colored. The subgraph in each round is chosen in nonincreasing weight order, such that its chromatic number is determined by the round number.
- First fit. Vertices are considered in decreasing order of weight and each vertex is assigned the first available color or interval.
- Best fit. Vertices are considered in reverse-perfect elimination order and each vertex is assigned the color class or interval it "fits" in best.
- Graph partitioning. Vertices are partitioned into groups with similar weight and we use the greedy coloring algorithm to coloring each subgraph with optimal number of colors. The interval assignment induced by this coloring is returned as the solution to the interval-coloring problem.

First fit and best fit are fairly standard heuristics for many resource allocation problems and have been analyzed extensively for problems, such as the bin-packing problem. Using old results and a few new observations, we point out that the first fit heuristic and the graph-partitioning heuristic provide an $O(\log n)$ approximation guarantee. The best-fit heuristic provides no such guarantee and we provide an example of a vertex-weighted interval graph for which the best-fit heuristic returns a solution to the max-coloring problem whose weight is $\Omega(\sqrt{n})$ times the weight of the optimal solution. In Pemmaraju and

Raman [2005], it is shown that GeomFit is a 4-approximation for max-coloring on perfect graphs. GeomFit as is, is known to do badly for interval-coloring, since OPT for max-coloring can be much larger than OPT for interval-coloring.

Our experiments show that, in general, GeomFit performs better than the rest of the heuristics and is typically very close to OPT, deviating by about 1.5% on average for max-coloring as well, as for interval-coloring. The first-fit heuristic also performs well, on average, deviating from OPT by about 6% for both problems. Best fit comes third and the graph-partitioning heuristic performs significantly worse than the others. Our basic data comes from about 10,000 runs of each of the three heuristics for each of the two problems on randomly generated chordal graphs of various sizes, sparsity, and structure.

Our experiments also reveal that best fit performs better on chordal graphs that are "irregular." Here, "regularity" refers to the variance in the sizes of maximal cliques—the greater this variance, the more irregular the graph.

2. THE ALGORITHMS

In this section we describe four simple algorithms for the interval-coloring and max-coloring problems.

2.1 Algorithm 1: GeomFit

This algorithm constructs a coloring in rounds, using a fresh set of colors in each round. In round i, a maximal c_i colorable subgraph of the graph is chosen in nonincreasing weight order, and colored with the fewest possible colors.

```
GeomFit(G, w)
    1. Let i = 0, l_i = 0
    2. While G ≠ φ do
            3. Set c_i = 2^i
            4. Let G_i = mkc(G, c_i)
            5. Color G_i optimally using colors l_i + 1, ..., l_i + c_i
            6. Set l_{i+1} = l_i + c_i, i = i + 1.
            7. Set G = G \ G_i.
    8. End While
```

A round of the algorithm corresponds to an iteration of the while loop. If the algorithm uses t rounds to color a graph G, then the rounds are numbered $0, 1, \ldots, t-1$. At each iteration, the algorithm calls the subroutine $mkc(G, c_i)$, that returns a maximal c_i-colorable subgraph of G, obtained by examining the vertices in nonincreasing order of weight. Here G is the subgraph of the input graph induced by the uncolored vertices and $c_i = 2^i$. When called, the subroutine $mkc(G, c_i)$ starts with an empty set S, and processes each vertex v of G, in nonincreasing order of weight. The subroutine tests if $G[S \cup \{v\}]$ is c_i-colorable or not and if it is, adds v to S, and proceeds with the next vertex of G. To perform this test, $mkc(G, c_i)$ calls the algorithm A that returns the minimum vertex coloring of G. Assuming that A runs in polynomial time, each call to the

subroutine $mkc(G, c_i)$ also runs in polynomial time. Hence, step 4 of `GeomFit` is also executed in polynomial time by calling algorithm A. Since the number of rounds $t = O(\log n)$, the entire algorithm runs in polynomial time. For perfect graphs, the algorithm A uses the ellipsoid method [Grötschel et al. 1993]. However, for chordal graphs, we can compute a minimum coloring in linear time using the perfect elimination order [Golumbic 1980].

For the interval-coloring problem, we return the solution induced by `GeomFit` for max-coloring; i.e., if a vertex v is assigned a color i by `GeomFit`, then we assign the interval $(\sum_{j=1}^{i-1} w(C_j), \sum_{j=1}^{i-1} w(C_j) + w(v))$ to v.

2.2 Algorithm 2: First Fit in Weight Order

For the interval-coloring problem, we preprocess the vertices and "round up" their weights to the nearest power of 2. Then, for both problems, we order the vertices of the graph in nonincreasing order of weights. Let $v_1, v_2, \ldots, v_n$ be this ordering. We process vertices in this order and use a "first-fit heuristic" to assign intervals and colors to vertices to solve the interval-coloring and max-coloring problem, respectively. We round up the weights to ensure an $O(\log n)$ approximation guarantee for interval coloring in Theorem 1.

The algorithm for interval coloring is as follows. To each vertex, we assign a real interval with nonnegative endpoints. To vertex v_1, we assign $(0, w(v_1))$. When we get to vertex $v_i, i > 1$, each vertex $v_j, 1 \leq j \leq i-1$ has been assigned an interval $I(v_j)$. Let U_i be the union of the intervals already assigned to neighbors of v_i. Then, $(0, \infty) - U_i$ is a nonempty collection of disjoint intervals. Because the weights are powers of 2 and vertices are considered in nonincreasing order of weights, every interval in $(0, \infty) - U_i$ has length at least $w(v_i)$. Of these, pick an interval $I = (a, b)$ with smallest right endpoint and assign the interval $(a, a + w(v_i))$ to v_i. This is $I(v_i)$.

For a solution to the max-coloring problem, we assume that the colors to be assigned to vertices are natural numbers and assign to each vertex v_i the smallest color not already assigned to a neighbor of v_i. We denote the two algorithms described above by `FFI` (short for first-fit for interval coloring) and `FFM` (short for first-fit for max-coloring), respectively.

We now observe that both algorithms provide an $O(\log(n))$-approximation guarantee. The following result is a generalization of the result from Chrobak and Ślusarek [1988].

THEOREM 1. *Let C be a class of graphs that is closed under duplication of vertices and suppose there is a function $\alpha(n)$ such that the first-fit on-line graph-coloring algorithm colors any n-vertex graph G in C with, at most, $\alpha(n) \cdot \chi(G)$ colors. Then, for any n-vertex graph G in C, the* `FFI` *algorithm produces a solution with span, at most, $2\alpha(n) \cdot OPT_I(G)$, where $OPT_I(G)$ is the optimal span of any feasible assignment of intervals to vertices.*

Perfect graphs are closed under vertex duplication. Since interval and chordal graphs are subclasses of perfect graphs, these graphs are closed under vertex duplication as well. The following is a generalization of the result from Pemmaraju et al. [2004a].

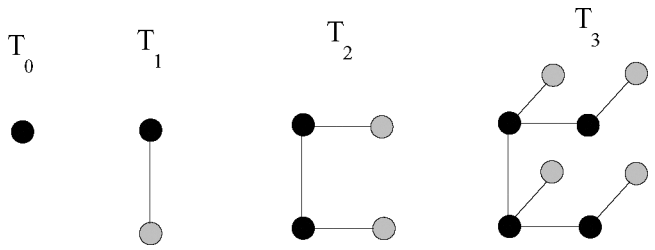

Fig. 1. The family of tight examples for FFI and FFM.

THEOREM 2. *Let C be a hereditary class of graphs*[1] *and suppose there is a function $\alpha(n)$ such that the first-fit on-line graph-coloring algorithm colors any n-vertex graph G in C with, at most, $\alpha(n) \cdot \chi(G)$ colors. Then, for any n-vertex graph G in C, the* FFM *algorithm produces a solution with weight, at most, $\alpha(n) \cdot OPT_M(G)$, where $OPT_M(G)$ is the optimal weight of any proper vertex coloring of G.*

Irani [1994] has shown that the first fit graph-coloring algorithm uses, at most, $O(\log(n)) \cdot \chi(G)$ colors for any n-vertex chordal graph G. This fact, together with the above theorems, implies that FFI and FFM provide $O(\log(n))$-approximation guarantees.

An example that is tight for both algorithms is easy to construct. Let $T_0, T_1, T_2, \ldots$ be a sequence of trees, where T_0 is a single vertex and T_i, $i > 0$, is constructed from T_{i-1} as follows. Let $V(T_{i-1}) = \{u_1, u_2, \ldots, u_k\}$. To construct T_i, start with T_{i-1} and add vertices $\{v_1, v_2, \ldots, v_k\}$ and edges $\{u_i, v_i\}$ for all $i = 1, 2, \ldots, k$. Thus, the leaves of T_i are $\{v_1, v_2, \ldots, v_k\}$ and every other vertex in T_i has a neighbor v_j for some j. Now consider a tree T_n in this sequence. Clearly, $|V(T_n)| = 2^n$. Assign to each vertex in T_n, a unit weight. To construct an ordering on the vertices of T_n, first delete the leaves of T_n. This leaves the tree T_{n-1}. Recursively construct the ordering on vertices of T_{n-1}, and prepend to this the leaves of T_n, in some order. It is easy to see that first-fit coloring algorithm that considers the vertices of T_n in this order uses n colors. As a result, both FFI and FFM have weight n, whereas OPT, in both cases, is 2. See Figure 1 for T_0, T_1, T_2, and T_3.

2.3 Algorithm 3: Best Fit in Reverse-Perfect Elimination Order

The third pair of algorithms that we experiment with are obtained by considering vertices in reverse-perfect elimination order and using a "best-fit" heuristic to assign intervals or colors. Let $v_1, v_2, \ldots, v_n$ be the reverse of a perfect elimination ordering of the vertices of G. Recall that if vertices are considered in reverse-perfect elimination order and colored, using the smallest color at each step, we get an optimal coloring of the given chordal graph. This essentially implies that the example of a tree with unit weights that forced FFI and FFM into worst-case behavior will not be an obstacle for this pair of algorithms.

[1]A class C of graphs is hereditary if $G \in C$ implies that every induced subgraph of G is also in C.

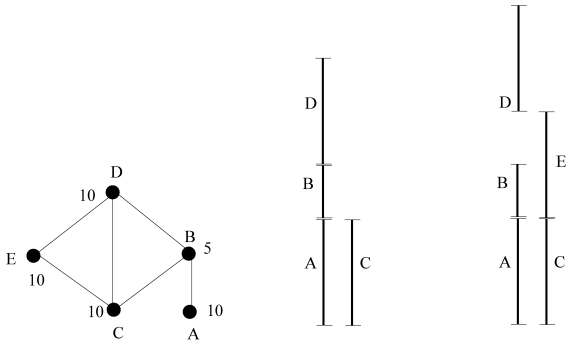

Fig. 2. The best-fit heuristic in action for interval coloring.

The algorithm for interval coloring is as follows. As before, to each vertex, we assign a real interval with non-negative endpoints and, to vertex v_1, we assign $(0, w(v_1))$. When we get to vertex v_i, $i > 1$, each vertex v_j, $1 \leq j \leq i-1$ has been assigned an interval $I(v_j)$. Let $M = |\cup_{j=1}^{i-1} I(v_j)|$ and let U_i be the union of the intervals $I(v_j)$, where $1 \leq j \leq i-1$ and v_j is a neighbor of v_i. If $U_i = (0, M)$, then v_i is assigned the interval $(M, M + w(v_i))$. Otherwise, if $U_i \neq (0, M)$, then $(0, M) - U_i$ is a nonempty collection of disjoint intervals. However, since the vertices were not processed in weight order, we are no longer guaranteed that there is any interval in $(0, M) - U_i$ with length at least $w(v_i)$. There are two cases.

Case 1. If there is an interval in $(0, M) - U_i$ of length at least $w(v_i)$, then pick an interval $I \in (0, M) - U_i$ of smallest length such that $|I| \geq w(v_i)$. Suppose $I = (a, b)$. Then assign the interval $(a, a + w(v_i))$to v_i.

Case 2. Otherwise, if all intervals in $(0, M) - U_i$ have length less than $w(v_i)$, pick the largest interval $I = (a, b)$ in $(0, M) - U_i$ (breaking ties arbitrarily) and assign $(a, a + w(v_i))$ to v_i. Note that this assignment of an interval to v_i causes the interval assignment to become infeasible. This is because there is some neighbor of v_i that has been assigned an interval with left endpoint b and $(a, a + w(v_i))$ intersects this interval. To restore feasibility, we lift all intervals "above" b by $\Delta = (a + w(v_i)) - b$. In other words, for every vertex v_j, $1 \leq j \leq i$, $I(v_j) = (c, d)$, if $c \geq b$, then set $I(v_j) = (c + \Delta, d + \Delta)$. It is easy to see that this restores feasibility to the interval assignment.

Best fit tries to minimize the increase in span of the intervals at each instance. Consider the chordal graph shown in Figure 2. The numbers next to vertices are vertex weights and the letters are vertex labels. The ordering of vertices A, B, C, D, E is a reverse-perfect elimination ordering. By the time we get to processing vertex E, the assignment of intervals to vertices is as shown in the middle in Figure 2. When E is processed, we look for "space" to fit it in and find the interval (10, 15), which is not large enough for E. So we move the interval $I(D)$ up by five units to make space for $I(E)$ and obtain the assignment shown in the figure on the right.

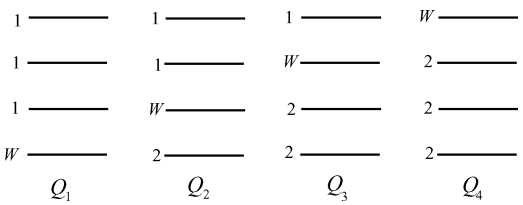

Fig. 3. A bad example for the best-fit heuristic.

A similar "best-fit" solution to the max-coloring problem is obtained as follows. Let k be the size of a maximum clique in G. Start with a palette of colors $C = \{1, 2, \ldots, k\}$ and an assignment of color 1 to vertex v_1. Let $AC(v_i) \subseteq C$ be the colors available for v_i. For each color j, let W_j denote the maximum weight among all vertices colored j; for an empty color class j, $W_j = 0$. Color vertex v_i with a color $j \in AC$ that maximizes W_j, with ties broken arbitrarily. This ensures that the color we assign to v_i minimizes the increase in the weight of the coloring.

We will call these "best-fit" algorithms for interval coloring and max-coloring, `BFI` and `BFM`, respectively.

An example that forces the best-fit algorithms, `BFI` and `BFM` to perform badly is the following. Consider a graph G with m disjoint cliques, each clique containing m vertices. Let the cliques be labeled $Q_1, Q_2, \ldots, Q_m$. For each i, $1 \leq i \leq m$, the distribution of weights of vertices in Q_i is as follows: there are $(i-1)$ vertices with weight 2, one vertex with weight W, and $(m-i)$ vertices with weights 1. Any ordering of vertices is a perfect elimination ordering of G. Thus, suppose that `BFM` processes vertices in the following order: vertices of Q_1, followed by vertices of Q_2, followed by vertices of Q_3, etc. The vertices of each Q_i are ordered as follows: vertices with weight 2 come first, followed by the vertex of weight W, followed by the vertices of weight 1. It is easy to check that if vertices are processed in this order, then `BFM` will produce a coloring with m color classes, such that each color class contains a vertex of weight W. This solution has weight $m \cdot W$ as compared to OPT, which has weight $W + 2(m-1)$. Thus, this is an example that forces `BFM` to produce a solution at least $\Omega(m)$ times OPT. This is $\Omega(\sqrt{n})$, since the number of vertices $n = m^2$. In Figure 3, this example is shown as a set of intervals with $m = 4$. The intervals correspond to vertices and pairwise intersection of intervals corresponds to edges. Each row of intervals corresponds to a color class chosen by `BFM`. The optimal coloring in this instance would put the intervals with weight W in one row.

2.4 Algorithm 4: Via Graph Partitioning

Another pair of algorithms for interval coloring and max-coloring can be obtained by partitioning the vertices of the given graph into groups with similar weight. Let W be the maximum vertex weight. Fix an integer $k \geq 1$ and partition the range $[1, W]$ into $(k+1)$ subranges:

$$\left[1, \frac{W}{2^k}\right], \left(\frac{W}{2^k}, \frac{W}{2^{k-1}}\right], \ldots, \left(\frac{W}{2^2}, \frac{W}{2}\right] \left(\frac{W}{2}, W\right].$$

For i, $1 \leq i \leq k$, let $R_i = (W/2^i, W/2^{i-1}]$ and let $R_{k+1} = [1, W/2^k]$. Partition the vertex set V into subsets V_i, $1 \leq i \leq (k+1)$ defined as $V_i = \{v \in V \mid w(v) \in R_i\}$. For each i, $1 \leq i \leq (k+1)$, let G_i be the induced subgraph $G[V_i]$. We ignore the weights and color each subgraph G_i with the fewest number of colors, using a fresh palette of colors for each subgraph G_i. For the max-coloring problem, we simply use this coloring as the solution.

For the interval-coloring problem, we turn the coloring into an assignment of intervals to vertices as follows. Let $Q_1, Q_2, \ldots, Q_t$ be color classes produced by the above algorithm. For each i, $1 \leq i \leq t$, let W_i be the maximum vertex weight of a vertex in Q_i. Let $W_0 = 0$. For each $v \in C_i$, we assign the interval $(\sum_{s=0}^{i-1} W_s, \sum_{s=0}^{i-1} W_s + w(v))$. This is an interval coloring of G because vertices in distinct color classes are assigned disjoint intervals. The span of this interval assignment is $\sum_{s=1}^{t} W_i$. This is identical to the total weight of the solution to max-coloring as well.

We will call these graph partitioning-based algorithms for interval coloring and max-coloring, `GPI` and `GPM`, respectively.

THEOREM 3. *If we set $k = 2\log(n)$, then* `GPI` *and* `GPM` *produce $(4 \cdot \log(n) + o(1))$-approximations to both the interval-coloring, as well as the max-coloring problems on perfect graphs.*

PROOF. For i, $1 \leq i \leq k$, let α_i be the weight of the heaviest clique in $G[V_i]$. Let $\chi_i = \chi(G[V_i])$. Clearly, $\alpha_i \geq \chi_i \cdot W/2^i$. Let `OPT` refer to the weight of an optimal max-coloring and let $\texttt{OPT}_i$ refer to the weight of an optimal max-coloring restricted to vertices in V_i. Note that $\texttt{OPT}_i \geq \alpha_i$. Since `GPM` colors each V_i with exactly χ_i colors and since the weight of each vertex in V_i is at most $W/2^{i-1}$, the weight of the coloring that `GPM` assigns to V_i is at most $\chi_i \cdot W/2^{i-1} \leq 2 \cdot \alpha_i \leq 2 \cdot \texttt{OPT}_i$. Since `GPM` uses a fresh palette of colors for each V_i, the weight of the coloring of $\cup_{i=1}^{k} V_i$ is, at most

$$2 \cdot \sum_{i=1}^{k} \texttt{OPT}_i \leq 2 \cdot \sum_{i=1}^{k} \texttt{OPT} = 4\log(n) \cdot \texttt{OPT}$$

Since $k = 2\log(n)$, $W/2^k = W/n^2$. Therefore, any coloring of V_{k+1} adds a weight of at most W/n to the coloring of the rest of the graph. Since $W \leq \texttt{OPT}$, `GPM` colors the entire graph with weight at most $(4 \cdot \log(n) + 1/n)\texttt{OPT}$.

The lower bound on α_i that was used in the above proof for max-coloring also applies to interval coloring and we get the same approximation factor for interval coloring. □

3. OVERVIEW OF THE EXPERIMENTS

3.1 How Chordal Graphs are Generated

We have implemented an algorithm that takes in parameters n (a positive integer) and α (a real number in $[0, 1]$) and generates a random chordal graph with n vertices, whose sparsity is characterized by α. The smaller the value of α, the more sparse the graph. In addition, the algorithm can run in two modes;

Table I. Properties of 20 Instances of Graphs with $n = 250$ and $\alpha = 0.9$[a]

	Mode 1									
No. of maximal cliques	149	126	110	126	147	116	119	119	128	149
Size of largest clique	13	14	12	12	14	11	12	12	12	14
Size of smallest clique	4	3	5	3	5	4	4	4	3	5
Mean clique size	8.58	7.35	8.35	7.83	9.41	7.51	7.10	7.31	7.98	9.53
Variance	4.06	3.83	3.23	2.35	3.32	1.99	2.36	2.82	3.61	3.23
	Mode 2									
No. of maximal cliques	220	216	216	218	218	219	213	216	219	219
Size of largest clique	29	33	33	31	14	31	30	36	30	30
Size of smallest clique	5	3	5	7	4	5	7	5	4	4
Mean clique size	20.13	22.34	22.37	24.00	21.15	21.83	25.17	23.70	22.80	20.89
Variance	28.43	30.02	30.69	29.55	33.98	31.73	48.72	32.66	25.81	29.68

[a]Ten of these were generated in Mode 1 and the other ten in Mode 2.

in Mode 1 it generates somewhat "regular" chordal graphs and in Mode 2 it generates somewhat "irregular" chordal graphs.

The algorithm generates chordal graphs with $n, (n-1), \ldots, 2, 1$ as a perfect elimination ordering. In the ith iteration of the algorithm, vertex i is connected to some subset of the vertices in $\{1, 2, \ldots, i-1\}$. Let G_{i-1} be the graph containing vertices $1, 2, \ldots, (i-1)$, generated after iteration $(i-1)$. Let $\{C_1, C_2, \ldots, C_t\}$ be the set of maximal cliques in G_{i-1}. It is well known that any chordal graph on n vertices has, at most, n maximal cliques. Thus, we explicitly maintain the list of maximal cliques in G_{i-1}. We pick a maximal clique C_j and a random subset $S \subseteq C_j$ and connect i to the vertices in S. This ensures that the neighbors of i in $\{1, 2, \ldots, i-1\}$ form a clique, thereby ensuring that $n, (n-1), \ldots, 2, 1$ is a perfect elimination ordering.

We use the parameter α in order to pick the random subset S. For each $v \in C_j$, we independently add v to set S with probability α. This makes the expected size of S equal $\alpha \cdot |C_j|$. The algorithm also has a choice to make on how to pick C_j. One approach is to choose C_j uniformly at random from the set $\{C_1, C_2, \ldots, C_t\}$. This is Mode 1 and it leads to "regular" random chordal graphs, that is, random chordal graphs in which the sizes of maximal cliques show small variance. Another approach is to choose a maximal clique with largest size from among $\{C_1, C_2, \ldots, C_t\}$. This is Mode 2 and it leads to more "irregular" random chordal graphs, that is, random chordal graphs in which there are a small number of very large maximal cliques and many very small maximal cliques. Graphs generated in the two modes seem to be structurally quite different. This is illustrated in Table I, where we show information associated with 10 instances of graphs with $n = 250$ and $\alpha = 0.9$ generated in Modes 1 and 2. Each column corresponds to one of the 10 instances and comparing corresponding Modes 1 and 2 rows easily reveals the fairly dramatic difference in these graphs. For example, the mean clique size in Mode 1 is about 8.5, while it is about 22 in Mode 2. Even more dramatic is the large difference in the variance of the clique sizes, which justifies our earlier observation that Mode 2 chordal graphs tend to have a few large cliques and many very small cliques relative to Mode 1 chordal graphs.

3.2 How Weights Are Assigned

Once we have generated a chordal graph G, we assign weights to the vertices as follows. This process is parameterized by W, the maximum possible weight of a vertex. Let k be the chromatic number of G and let $\{C_1, C_2, \ldots, C_k\}$ be a k-coloring of G. Since G is a chordal graph, it contains a clique of size k. Let $Q = \{v_1, v_2, \ldots, v_k\}$ be a clique in G with $v_i \in C_i$. For each v_i, pick $w(v_i)$ uniformly at random from the set of integers $\{1, 2, \ldots, W\}$. Thus, the weight of Q is $\sum_{i=1}^{k} w(v_i)$. For each vertex $v \in C_i - \{v_i\}$, pick $w(v)$ uniformly at random from $\{1, 2, \ldots, w(v_i)\}$. This ensures that $\{C_1, C_2, \ldots, C_k\}$ is a solution to max-coloring with weight $\sum_{i=1}^{k} w(v_i)$ and the interval assignment induced by this coloring is an interval coloring of span $\sum_{i=1}^{k} w(v_i)$. Since $\sum_{i=1}^{k} w(v_i)$ is also the weight of the clique Q, which is a lower bound on OPT in both cases, we have that OPT $= \sum_{i=1}^{k} w(v_i)$ in both cases. The advantage of this method of assigning weights is that it is simple and gives us the value of OPT for both problems. The disadvantage is that, in general, OPT for both problems can be strictly larger than the weight of the heaviest clique and thus, by generating only those instances for which OPT equals the weight of the heaviest clique, we might be missing a rich class of problem instances. Thus, we additionally tested our algorithms on instances of chordal graphs for which the weights were assigned at random. For these instances, we use the maximum weight clique as a lower bound for OPT and, as a consequence, the deviations reported for the algorithms are an overestimate of the real deviations.

3.3 Main Observations

For our main experiment we generated instances of random chordal graphs with number of vertices $n = 10, 20, 30, \ldots, 550$. For each value of n, we used values of $\alpha = 0.1, 0.2, \ldots, 0.9$. For each of the 55×9 (n, α) pairs, we generated 10 random vertex-weighted chordal graphs in Mode 1 and 10 random vertex-weighted chordal graphs in Mode 2. The vertex weights are assigned as described above, with the maximum weight W fixed at 1000. We ran each of the four heuristics for each of the two problems and averaged the weight and span of the solutions over the 10 instances for each (n, α) pairs separately for Modes 1 and 2 graphs. Thus each heuristic was evaluated on 4950 instances of each mode, for each problem. We then generated the same number of instances, with n, α, and the modes varying just as before, but this time assigning to each vertex, a weight chosen uniformly at random from $[0, 1000]$. We repeated all eight algorithms on these random instances and used the maximum-weighted clique as a lower bound for OPT.

For the max-coloring problem on interval graphs, the gap between the maximum-weight clique and OPT can be unbounded. We can construct examples where the gap is as large as $\Omega(\log n)$ Pemmaraju S. V. et al. [2004a]. For the interval-coloring problem, the gap between OPT and the maximum-weight clique is known to be a constant, while for chordal graphs it is unknown. This gap may affect the reported deviations in ways that are hard to determine.

Table II. This Shows Aggregate Performance of 4950 Runs of the Four Heuristics for Max-coloring Separately for Modes 1 and 2 Graphs[a]

	Mode 1				Mode 2			
	BFM	FFM	GPM	GFM	BFM	FFM	GPM	GFM
Equals OPT	936	3170	0	3580	3693	3451	0	3820
Equals χ	4950	3539	0	4019	4950	3451	0	3820
% Deviation	14.40	1.627	58.26	1.31	2.45	1.94	29.99	1.53
% Max Deviation	157.12	30.38	127.67	30.38	54.99	28.88	82.78	28.88
Equals LB (R)	90	111	0	121	101	111	0	121
Equals χ(R)	4950	2712	20	3232	4950	588	0	707
% Deviation (R)	24.79	17.08	74.69	16.08	24.90	12.88	44.08	12.74
% Max Deviation (R)	74.88	62.52	129.8	47.03	141.16	63.29	122.9	63.29

[a]The first four rows show the case when the weights are chosen so that OPT equals the weight of the maximum weight clique. The next four rows show the case when the weights are chosen randomly. In this case, the algorithms are compared against the weight of the maximum-weight clique. The row "Equals OPT" lists the number of times each heuristic produces a coloring with weight equal to OPT, the row "Equals χ" lists the number of times each heuristic produces a coloring using minimum number of colors, and the row "Deviation" lists the percentage deviation of the weight of the solution produced from OPT or the lower bound on OPT, averaged over the 4950 runs. The row "Max Deviation" lists the maximum percentage deviation of the four algorithms from OPT or the lower bound.

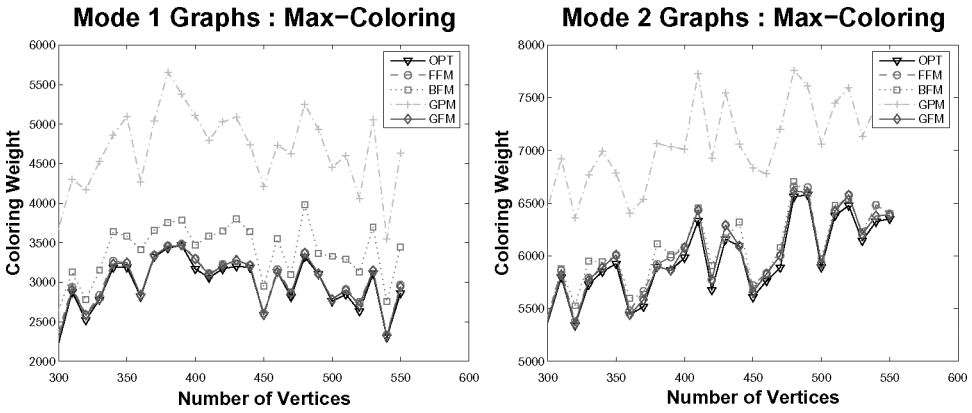

Fig. 4. Graphs showing values for max-coloring Modes 1 and 2 chordal graphs. The x axis corresponds to the number of vertices in the graph; the y axis corresponds to the max-color weight.

3.3.1 *Max-coloring.* The data for the max-coloring problem is presented in the following tables[2] and graphs. Table II summarizes the performance of the four heuristics for the max-coloring problem for both Modes 1 and 2 chordal graphs. Graphs showing the performance of the four heuristics for the max-coloring problem on Modes 1 and 2 chordal graphs are shown in Figures 4 and 5.

Based on our results we make the following observations regarding the max-coloring problem.

[2]All the raw data and code for the experiments is available at http://www.cs.uiowa.edu/∼rraman/chordalExp.html.

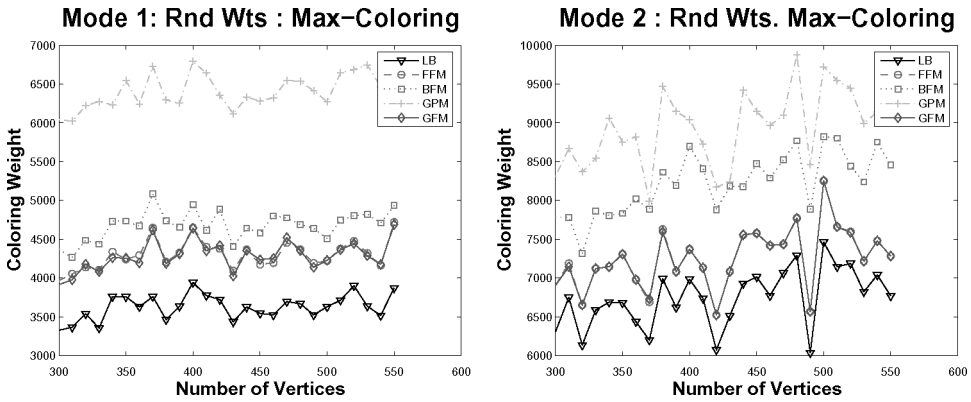

Fig. 5. Graphs showing values for max-coloring Modes 1 and 2 chordal graphs with randomly assigned weights. The x axis corresponds to the number of vertices in the graph; the y axis corresponds to the max-color weight.

1. Algorithm `GeomFit` consistently outperforms other algorithms for max-coloring, deviating only by about 1.5%, on average, for both Modes 1 and 2 graphs. Although the constant-factor guarantee for `GeomFit`, holds only in the worst case, this by itself does not give much indication regarding the performance of `GeomFit`, in practice. The fact that `GeomFit` works so well even in practice is a vindication of efforts by theoreticians to design algorithms with good provable approximation guarantees.
2. Although first fit's performance is not as good as `GeomFit`, its performance is only slightly worse than `GeomFit`, deviating only about 2% on both Modes 1 and 2 graphs.
3. In the case with random weights, although the average deviation of `GeomFit` and first-fit improves as we move from Mode 1 to Mode 2 graphs, the maximum percentage deviation deteriorates significantly for `GeomFit` and the maximum percentage deviation deteriorates only slightly for first-fit. However, the performance of first fit is almost as good as `GeomFit` for Modes 1 and 2 graphs with random weights.
4. The best-fit heuristic seems to be at a disadvantage, because it is constrained to always use as many colors as the chromatic number. In general, it does worse than the first-fit heuristic. However, the performance of best-fit improves as we move from Modes 1 to 2 graphs. The first-fit and `GeomFit` heuristics use more colors than the chromatic number a fair number of times—28 and 30% of the time for Modes 1 and 2 graphs, respectively. The implementation of best-fit here differs from the conference version in two respects. First, in the conference version of the paper [Pemmaraju et al. 2004b], we used the same perfect elimination scheme used to compute the weights of the vertices in Modes 1 and 2 graphs, while now, we randomly reorder the vertices and compute a fresh perfect elimination order and use this to compute a best fit. This explains the increased deviation of best fit for Mode 2 graphs compared to the conference version. The second difference is that in the conference version, to find a best fit interval for a vertex v, we were not

Table III. Aggregate Performance of 4950 Runs of the Four Heuristics for Interval Coloring, Separately for Modes 1 and 2 Graphs[a]

	Mode 1				Mode 2			
	BFI	FFI	GPI	GFI	BFI	FFI	GPI	GFI
Equals OPT	2959	3330	1874	4450	2738	1971	240	3820
% Deviation	7.52	2.63	7.99	0.399	5.64	5.95	14.76	1.54
% Max Deviation	75.72	28.15	45.67	24.64	50.47	54.45	52.32	28.89
Equals LB (R)	1820	1741	958	1882	407	291	91	272
% Deviation (R)	22.75	11.99	14.75	7.34	18.11	11.26	23.76	11.72
% Max Deviation (R)	191.13	52.94	57.23	43.24	102	52.14	70.04	52.14

[a]The first three rows are for the case when the weights are chosen so that OPT is equal to the maximum-weight clique; the next three rows are the case when the weights are chosen randomly. For the case with random weights, the performance of the algorithms is compared against the weight of the maximum-weight clique. The row "Equals OPT" lists the number of times each heuristic produces a coloring with weight equal to OPT, the row "% Deviation" lists the percentage deviation of the weight of the solution produced from OPT or the lower bound on OPT, averaged over the 4950 runs, and the row "Max Deviation" lists maximum percentage deviation of any algorithm from OPT or the lower bound.

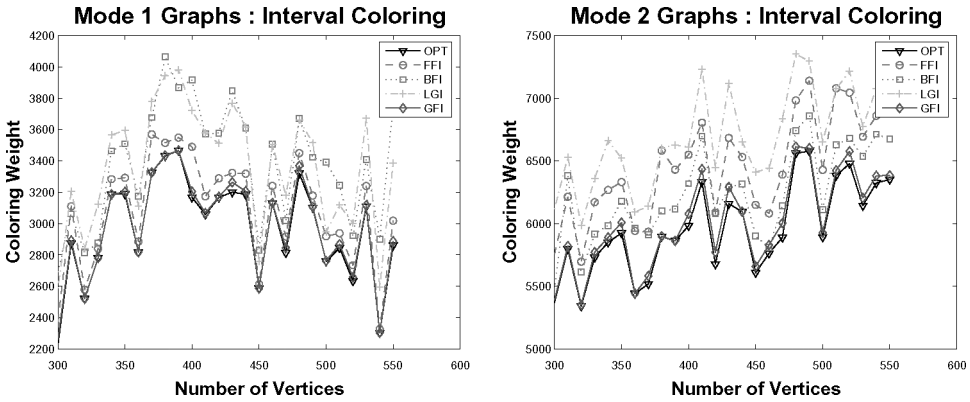

Fig. 6. Graph showing values for interval-coloring Modes 1 and 2 chordal graphs. The x axis corresponds to the number of nodes in the graph; the y axis corresponds to the interval color weight.

testing the interval $[0, l_u)$, where l_u is the lowest left end-point of any interval assigned to a neighbor of v. This explains the improved performance of best fit on Mode 1 graphs.

5. The graph-partitioning heuristic is not competitive at all, relative to best fit and first fit, in any of the cases, despite the $O(\log(n))$-factor approximation guarantee it provides.

3.4 Interval Coloring

We now present the data for the interval-coloring problem on chordal graphs. We summarize our results in Table III , which shows the average deviation of the four algorithms over all the runs, for both modes. The graphs showing the performance of the algorithms are presented in Figures 6 and 7. We make the following observations.

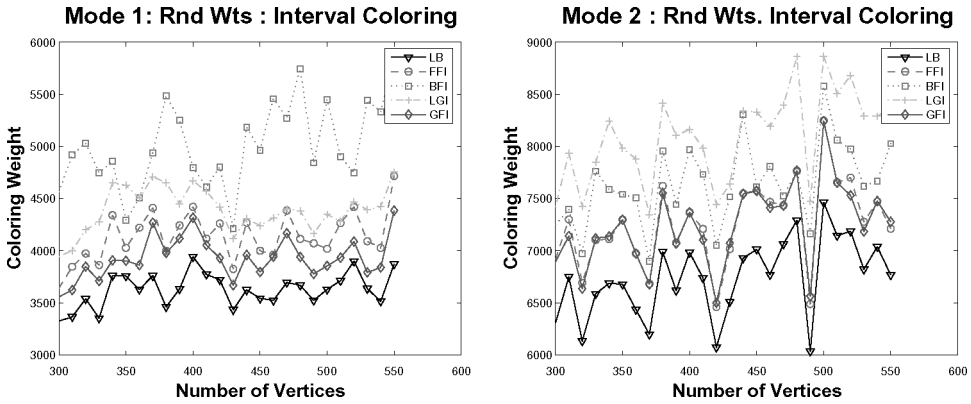

Fig. 7. Graph showing values for interval coloring Modes 1 and 2 chordal graphs with randomly assigned weights. The x axis corresponds to the number of nodes in the graph; the y axis corresponds to the interval color weight.

1. For Modes 1 and 2 graphs, where the value of OPT is known, `GeomFit` consistently outperforms the other algorithms for interval-coloring, deviating only about 1.5%, on average.
2. Mode 2 graphs are significantly harder for all heuristics, except best fit. Although all heuristics equal OPT fewer times on Mode 2 than on Mode 1 graphs, the average and maximum percentage deviations for best fit improve as we move from Mode 1 to Mode 2 graphs, in contrast with the other three heuristics.
3. For Modes 1 and 2 graphs with random weights, however, the situation is less clear. Although `GeomFit` performs better than the other algorithms for Mode 1 graphs with random weights, first fit has the smallest average deviation for Mode 2 graphs with random weights. Furthur, best fit equals OPT more often than any other heuristic for Mode 2 graphs with random weights.
4. The weights in first fit were rounded up to the nearest power of 2 in order to ensure the $O(\log n)$ bound on the approximation factor. In fact, there are simple examples where first fit can be made to perform worse if the weights of the vertices are not raised to the nearest power of 2. However, in practice, the performance of first fit improves for interval coloring of Mode 2 graphs if we use the original weights. The deviation is, at most, 0.54% on Mode 1 graphs and at most 1.39% for Mode 2 graphs, compared to 2.63 and 5.95% for Modes 1 and 2 graphs, respectively, with the weights rounded up to the nearest power of 2.
5. For the case with random weights as well, the performance of best fit improves as we move from Modes 1 to 2 graphs.

4. CONCLUSION

Our goal was to evaluate algorithms for max-coloring and interval coloring of chordal graphs. For both problems, the new `GeomFit` heuristic works as well,

if not better, than first fit. The other heuristics seem to be significantly worse for both problems. More generally, our experiments indicate that for the max-coloring problem, GeomFit, first fit, best fit, and graph-partitioning is the ordering of the algorithms in worsening order of performance. However, for the interval-coloring problem, both GeomFit and first fit perform better than the other heuristics. While GeomFit performs better on Mode 1 graphs, first fit performs better on Mode 2 graphs with random weights. From an implementation point of view, first fit may be better than GeomFit. First-fit is a simple heuristic to implement, irrespective of the underlying graph structure. GeomFit relies on the procedure mkc(G, c_i) to compute the optimum coloring of a graph. For classes of graphs that have less structure than chordal graphs, there is typically no efficient algorithm to find an optimal coloring or even approximate it. For example, the only known algorithm to compute a coloring of a perfect graph uses the ellipsoid algorithm of Grötschel et al. [1993] and this is prohibitively expensive, in practice. However, for special classes of perfect graphs, like chordal and interval graphs, we can compute a minimum coloring efficiently. Best fit exhibits a performance that is different from the other algorithms in going from Modes 1 to 2 graphs. While the performance of the three heuristics becomes worse as we move from Modes 1 to 2 graphs, the performance of best fit improves. This may suggest best fit as a candidate of choice for highly irregular graphs, although this issue needs further exploration. The performance of GeomFit for interval coloring on instances of Modes 1 and 2 chordal graphs is surprisingly good, leading us to conjecture that an algorithm similar to GeomFit might, indeed, lead to a constant-factor approximation algorithm for interval-coloring, as well.

REFERENCES

BAR-NOY, A., HALLDORSSON, M., KORTSARZ, G., SALMAN, R., AND SHACHNAI, H. 2000. Sum multicoloring of graphs. *Journal of Algorithms 37*, 2, 422–450.

BUCHSBAUM, A. L., KARLOFF, H., KENYON, C., REINGOLD, N., AND THORUP, M. 2003. OPT versus LOAD in dynamic storage allocation. In *Proceedings of the 35th Annual ACM Symposium on Theory of Computing (STOC)*. 632–646.

CHROBAK, M. AND ŚLUSAREK, M. 1988. On some packing problems related to dynamic storage allocation. *Informatique théorique et Applications/Theoretical Informatics and Applications 22*, 4, 487–499.

CONFESSORE, G., DELL'OLMO, P., AND GIORDANI, S. 2002. An approximation result for the interval coloring problem on claw-free chordal graphs. *Discrete Applied Mathematics 120*, 71–88.

FABRI, J. 1979. Automatic storage optimization. *ACM SIGPLAN Notices: Proceedings of the ACM SIGPLAN '79 on Compiler Construction 14*, 8, 83–91.

GAREY, M. R. AND JOHNSON, D. S. 1979. *Computers and Intractability: A Guide to the theory of NP-completeness*. Freeman, San Fransisco, CA.

GERGOV, J. 1996. Approximation algorithms for dynamic storage allocation. In *Proceedings of the 4th European Symposium on Algorithms: Lecture Notes in Computer Science 1136*. 52–61.

GERGOV, J. 1999. Algorithms for compile-time memory optimization. In *Proceedings of the 10th ACM-SIAM Symposium on Discrete Algorithms*. S907–S908.

GOLUMBIC, M. C. 1980. *Algorithmic Graph Theory and Perfect Graphs*. Academic Press, New York.

GOVINDARAJAN, R. AND RENGARAJAN, S. 1996. Buffer allocation in regular dataflow networks: An approach based on coloring circular-arc graphs. In *Proceedings of the 2nd International Conference on High Performance Computing*.

GRÖTSCHEL, M., LOVÁSZ, L., AND SCHRIJVER, A. 1993. *Geometric Algorithms and Combinatorial Optimization*. Springer-Verlag, New York.

IRANI, S. 1994. Coloring inductive graphs on-line. *Algorithmica 11*, 1, 53–72.

KIERSTEAD, H. A. 1988. The linearity of first-fit coloring of interval graphs. *SIAM J. Discrete Math 1*, 526–530.

KIERSTEAD, H. A. 1991. A polynomial time approximation algorithm for dynamic storage allocation. *Discrete Mathematics 88*, 231–237.

KIERSTEAD, H. A. AND TROTTER, W. T. 1981. An extremal problem in recursive combinatorics. *Congressus Numerantium 33*, 143–153.

PEMMARAJU, S. V., RAMAN, R., AND VARADARAJAN, K. 2004a. Buffer minimization using max-coloring. In *Proceedings of The ACM-SIAM Symposium on Discrete Algorithms (SODA)*. 562–571.

PEMMARAJU, S. V., PENUMATCHA, S., AND RAMAN, R. 2004b. Approximating interval coloring and max-coloring in chordal graphs. In *Proceedings of The Third Workshop on Efficient and Experimental Algorithms (WEA)*. 399–416.

PEMMARAJU, S. V. AND RAMAN, R. 2005. Approximation algorithms for the max-coloring problem. In *International Colloquium on Automata, Languages and Computation (ICALP)*. 1064–1075.

RUS, T. AND PEMMARAJU, S. V. 1997. Using graph coloring in an algebraic compiler. *Acta Informatica 34*, 3, 191–209.

TARJAN, R. E. AND YANNAKAKIS, M. 1984. Simple linear-time algorithms to test chordality of graphs, test acyclicity of hypergraphs, amd selectively reduce acyclic hypergraphs. *SIAM Journal on Computing 13*, 566–579.

Received February 2005; revised December 2005; accepted June 2006