Approximating Interval Coloring and Max-Coloring in Chordal Graphs

Sriram V. Pemmaraju, Sriram Penumatcha, and Rajiv Raman

The Department of Computer Science, The University of Iowa, Iowa City, IA 52240-1419,
e-mail : sriram@cs.uiowa.edu, spenumat@cs.uiowa.edu, rraman@cs.uiowa.edu

Abstract. 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 1970's. The max-coloring problem models the problem of minimizing the total buffer size needed by a dedicated memory manager. This problem arises whenever there is a need to design dedicated memory managers that provide better performance than the general purpose memory management of the operating system. Both problems are NP-complete even for interval graphs, though there are constant-factor approximation algorithms for both problems on interval graphs.

In this paper we consider these problems for *chordal graphs*. These graphs naturally generalize interval graphs and can be defined as the class of graphs that have no induced cycle of length > 3. There are no known constant-factor approximation algorithms for either interval coloring or for max-coloring on chordal graphs. However, we point out in this paper that there are several simple $O(\log(n))$-factor approximation algorithms for both problems. We experimentally evaluate and compare three simple heuristics: first-fit, best-fit, and a heuristic based on partitioning the graph into vertex sets of similar weight. Our experiments show that in general first-fit performs better than the other two heuristics and is typically very close to OPT, deviating from OPT by about 6% in the worst case for both problems. Best-fit provides some competition to first-fit, but the graph partitioning heuristic performs significantly worse than either. Our basic data comes from about 10000 runs of the each of the three heuristics for each of the two problems on randomly generated chordal graphs of various sizes and sparsity.

Our experiments also reveal an interesting trend – as the chordal graphs become more irregular, for both problems the performance of best-fit

improves, whereas the performance of first-fit worsens. Our overall conclusion is that first-fit is a practical, easy to implement heuristic, yielding close to optimal performance, even for applications in which the underlying graph is more complicated than an interval graph.

1 Introduction

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: (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 $|\cup_v I(v)|$. The interval coloring problem has a fairly long history dating back, at least to the 70's. 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 [5]). The main application of the interval coloring problem is in the *compile-time memory allocation problem.* Fabri [4] made this connection in 1979. In order to reduce the total memory consumption of source-code objects (simple variables, arrays, 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 [10] gave the first constant-factor approximation algorithm – an 80-approximation, that was improved by Kierstead himself to a 6-approximation [11]. This was followed by papers by Gergov [6, 7] who improved the approximation factor to 3. Recently, Buchsbaum et.al. [1] gave a $(2 + \varepsilon)$-algorithm for the problem.

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 needed for memory management in different applications. For example, [8] uses max-coloring to minimize buffer size in digital signal processing applications. In [13], 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.

[13] 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 [12] on on-line coloring interval graphs, they obtain a 2-approximation algorithm for interval graphs and a 3-approximation algorithm for circular arc graphs.

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 [13] 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 [1].

In general, algorithms for max-coloring can be used for interval coloring with minor modifications to make the interval assignment more "compact." These

connections motivate us to study interval coloring and max-coloring in the same framework.

Chordal graphs. For both the interval coloring and max-coloring problems, the assumption that the underlying graph is an interval graph is somewhat restrictive since 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.

The approximability of interval coloring and max-coloring on chordal graphs is not very well understood yet. As we point out in this paper, there are several $O(\log(n))$-factor approximation algorithms for both problems on chordal graphs, however the existence of constant-factor approximation algorithms for these problems is open.

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 [15] 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 and max-coloring interval graphs can be extended to chordal graphs. 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 [14]. 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, [3] shows a 2-approximation algorithm for the interval coloring problem on claw-free chordal graphs, leaving the problem open for chordal graphs in general.

The rest of the paper. In this paper, we consider three simple heuristics for the interval coloring and max-coloring problems and experimentally evaluate their performance. These heuristics are:

- **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 color 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.

Our experiments show that in general first-fit performs better than the other two heuristics and is typically very close to OPT, deviating from OPT by about 6% in the worst case for both problems. Best-fit provides some competition to first-fit, but the graph partitioning heuristic performs significantly worse than either. Our basic data comes from about 10000 runs of the each of the three heuristics for each of the two problems on randomly generated chordal graphs of various sizes and sparsity.

Our experiments also reveal that best-fit performs better on chordal graphs that are "irregular", while the performance of first-fit deteriorates slightly. In all other cases, first-fit is the best algorithm. Here, "regularity" refers to the variance in the sizes of maximal cliques – greater this variance, more irregular the graph.

2 The Algorithms

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

2.1 Algorithm 1: 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 non-increasing 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.

The algorithm for interval coloring is as follows. To each vertex we assign a real interval with non-negative 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 non-empty collection of disjoint intervals. Because the weights are powers of 2 and vertices are considered in non-increasing 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 by weight order for interval coloring) and `FFM` (short for first-fit by weight order 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 [2].

Theorem 1. *Let C be a class of graphs 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.*

The following is a generalization of the result from [13].

Theorem 2. *Let C be a class of graphs 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 of vertex coloring of G.*

Irani [9] 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 cost n, whereas OPT in both cases is 2. See Figure 1 for T_0, T_1, T_2, and T_3.

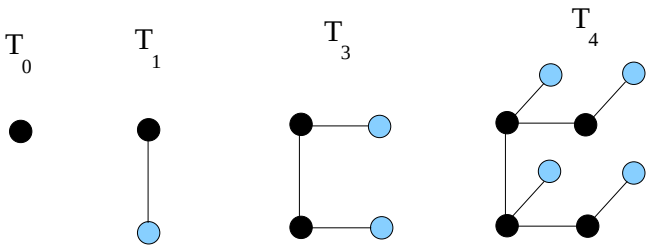

Fig. 1. The family of tight examples for `FFI` and `FFM`.

2.2 Algorithm 2: Best-fit in reverse perfect elimination order

A second 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.

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 non-empty 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 increase the endpoints of all intervals "above" b by $\Delta = (a + w(v_i)) - b$. In other words, for every vertex v_j, $1 \leq j \leq i$, if $I(v_j) = (c, d)$, where $c \geq b$ then $I(v_j)$ is reset to the interval $(c + \Delta, d + \Delta)$.

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 5 units to make space for $I(E)$ and obtain the assignment shown on the right.

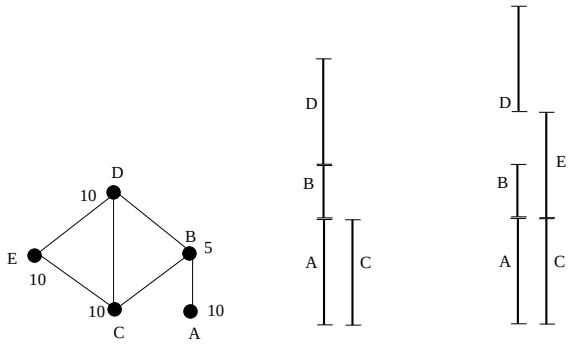

Fig. 2. The best-fit heuristic in action for interval coloring.

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 pallete 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, $W_j = 0$. From the subset of $AC(v_i)$ of available colors whose weights are atleast as large as $w(v_i)$, pick the color class j for v_i, whose weight is the smallest. If no such color exists, color vertex v_i with a color $j \in AC(v_i)$ for which W_j is maximum, 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 algorithm, `BFM` to perform badly is the following. Consider a graph G with n disjoint cliques, each clique containing n vertices. Let the cliques be labeled $C_1, C_2, \ldots, C_n$. For each i, $1 \leq i \leq n$, the distribution of weights of vertices in C_i is as follows: there are $(i-1)$ vertices with weight 2, one vertex with weight W, and $(n-i)$ vertices with weight 1. Here $W > 2$ is some integer. Any ordering of vertices is a perfect elimination ordering of G. So suppose that `BFM` processes vertices in the following order: vertices of C_1, followed by vertices of C_2, followed by vertices of C_3, etc. The vertices of each C_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 n color classes, such that each color class contains a vertex of weight W. This solution

has weight $n \cdot W$ as compared to OPT which has weight $W + 2(n-1)$ and thus this is an example that forces BFM to produce a solution at least $\Omega(n)$ times OPT. In Figure 3, this example is shown as a set of intervals with $n = 4$. 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.

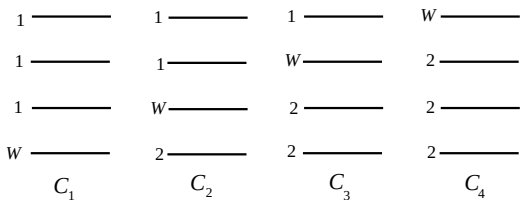

Fig. 3. A bad example for the best-fit heuristic.

2.3 Algorithm 3: 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 $1 \leq k \leq \log_2 W$ 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 pallette of colors for each subgraph G_i. For the max-coloring problem, we simply use this coloring as the solution. The solution to the interval coloring problem is simply the interval assignment induced by the coloring.

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.*

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 $\chi_i \cdot W/2^{i-1} \leq 2 \cdot \alpha_i \leq 2 \cdot \mathtt{OPT}_i$. Since GPM uses a fresh pallette of colors for each V_i, the weight of the coloring of $\cup_{i=1}^{k} V_i$ is at most

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

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

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; 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. So 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 aproach 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 small maximal cliques.

Graphs generated in the two modes seem to be structurally quite different. This is illustrated in the table in Figure 4, where we show information associated with 10 instances of graphs with $n = 250$ and $\alpha = 0.9$ generated in mode 1 and in mode 2. Each column corresponds to one of the 10 instances and comparing corresponding mode 1 and mode 2 rows easily reveals the 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 and this justifies our earlier observation that mode 2 chordal graphs tend to have a few large cliques and many small cliques, relative to mode 1 chordal graphs.

	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

Fig. 4. Properties of 20 instances of graphs with $n = 250$ and $\alpha = 0.9$. Ten of these were generated in Mode 1 and the other ten in mode 2.

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 paramaterized 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.

We also tested our algorithms on instances of chordal graphs for which the weights were assigned uniformly at random. For these algorithms, we use the maximum weighted clique as a lower bound for OPT.

Table 1. Results of our main experiments on mode 1 random chordal graphs, evaluating heuristics for the max-coloring problem.

$\|V\|$	$\|E\|$	OPT	Best Fit	First Fit	Partition	$\frac{(BF-OPT)}{OPT}$	$\frac{(FF-OPT)}{OPT}$	$\frac{(GPM-OPT)}{OPT}$
40	64.4333	2292.16	2416.64	2309.09	3184.84	5.43108	0.738751	38.9454
80	139.178	2474.74	2604.09	2525.5	3569.16	5.22658	2.05094	44.2232
120	223.756	2605.77	2827.46	2660.5	3956.09	8.50763	2.10047	51.8205
160	304.822	2682.98	2941.02	2733.71	4095.01	9.61784	1.89093	52.6293
200	394.556	2712.4	2938.49	2761.17	4107.06	8.33538	1.79792	51.4178
240	451.978	2835.52	3161.7	2882.82	4394.1	11.5033	1.66812	54.9662
280	525.389	2863.43	3145.16	2928.19	4491.21	9.83862	2.26147	56.8471
320	626.744	2775.11	3046.61	2819.38	4335.47	9.78339	1.59513	56.2268
360	715.389	2913.18	3166.28	2992.4	4667.89	8.68811	2.71944	60.2336
400	821.822	3150.07	3427.57	3197.59	5033.11	8.80934	1.50861	59.7779
440	894.922	2983.16	3292.34	3065.1	4864.44	10.3645	2.7469	63.0637
480	973.078	3060.02	3395.98	3117.88	4841.14	10.9789	1.89069	58.2062
520	1061.74	3053.21	3393.28	3120.32	4883.03	11.138	2.19805	59.9311

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. We ran each of the three heuristics for the two problems and averaged the weight and span of the solutions over the 10 instances for each (n, α) pairs. Thus each heuristic was evaluated on 4950 instances, for each problem. The vertex weights are assigned as described above, with the maximum weight W fixed at 1000. We first conducted this experiment for the max-coloring problem on mode 1 and mode 2 chordal graphs and and then repeated them for the interval coloring problem.

We then generated the same number of instances, but this time assigning to each vertex, a weight chosen uniformly from $[0, 1000]$. We repeated each of the three heuristics for the two problems on these randomly generated instances. For these instances, we used the maximum weight clique as a lower bound to OPT. Note that the maximum weight clique can be quite small compared to the value of OPT.

Table 2. Results of our main experiments on mode 2 random chordal graphs, evaluating heuristics for the max-coloring problem.

$\|V\|$	$\|E\|$	OPT	Best Fit	First Fit	Partition	$\frac{(BF-OPT)}{OPT}$	$\frac{(FF-OPT)}{OPT}$	$\frac{(GPM-OPT)}{OPT}$
40	109.689	3094.16	3095.61	3151.24	3848.27	0.0470421	1.84506	24.3721
80	302.489	3927.78	3938.29	4023.13	5035	0.26761	2.42772	28.1895
120	524.8	4459.87	4471.22	4614.27	5758.13	0.254616	3.46199	29.11
160	799.044	4891.27	4903.94	5058	6387.23	0.259192	3.4088	30.5844
200	1052.07	5290.47	5293.46	5457.12	6976.49	0.0564958	3.15011	31.8691
240	1363.43	5390.86	5398.32	5556.09	7109.43	0.138506	3.06507	31.8795
280	1655.84	5749.18	5763.03	5937.57	7567.82	0.241001	3.2768	31.6331
320	1953.24	5776.97	5779.74	5982	7655.43	0.0480837	3.54915	32.5165
360	2261.96	5899.47	5916.26	6165.27	7847.38	0.284583	4.50549	33.0184
400	2639.39	5987.34	5992	6191.81	7905.62	0.0777566	3.41498	32.0389
440	2956.79	6161.91	6167.82	6390.23	8192.71	0.0959298	3.70538	32.9573
480	3243.42	6236.88	6251.6	6471.51	8228.43	0.236051	3.76203	31.9319
520	3645.54	6296.03	6302.76	6558.19	8388.33	0.106769	4.16382	33.232

Our data is presented in the following tables.[1] First we have two tables for our experiments for the max-coloring problem. The first two tables (Figures 1 and 2) for mode 1 and mode 2 chordal graphs respectively. This is followed by the table in Figure 5 that summarizes the performance of the three heuristics for the max-coloring problem for both mode 1 and mode 2 chordal graphs over all the runs. After this we present three tables (Figures 3, 4, and 6) that contains corresponding information for the interval coloring problem. The data for the experiments on graphs with randomly assigned vertex weights is presented in the appendix. Based on all this data, we make 4 observations.

1. First fit in decreasing order of weights is clearly a heuristic that returns solutions very close to OPT for both problems. Overall percentage deviations from OPT, each being over 4950 runs are 1.93, 4.47, 2.29 and 9.52 - the first two are for max-coloring on mode 1 and mode 2 chordal graphs respectively, and the next two are for interval coloring on mode 1 and mode 2 graphs. Even in the experiments on graphs with randomly assigned vertex weights, first-fit performs better than the other algorithms overall. The average deviations being less than 26% from the maximum weight clique for both problems (Figures 5 and 6 show the performance of the algorithms averaged over all the runs. Figures 4 and 5 in the appendix give the performance of the algorithms where vertex weights were assigned randomly). Note that in these cases, the percentage deviations are an exaggeration of the actual amount, since the maximum weight clique can be quite small compared to OPT.

[1] In each table, we only show representative values due to lack of space. The complete data is available at `http://www.cs.uiowa.edu/~rraman/chordalGraphExperiments.html`.

Table 3. Results of our main experiments on mode 1 random chordal graphs, evaluating heuristics for the interval coloring problem.

$\|V\|$	$\|E\|$	OPT	Best Fit	First Fit	Partition	$\frac{(BFI-OPT)}{OPT}$	$\frac{(FFI-OPT)}{OPT}$	$\frac{(GPI-OPT)}{OPT}$
40	64.4333	2292.16	2377.64	2336.58	2443.47	3.72963	1.93801	6.60126
80	139.178	2474.74	2612.68	2531.07	2709.23	5.57364	2.27588	9.47528
120	223.756	2605.77	2803.87	2680.47	2839.19	7.60237	2.86672	8.95791
160	304.822	2682.98	3032.76	2767.01	2957.28	13.0369	3.13209	10.2237
200	394.556	2712.4	3055.42	2808.32	2975.33	12.6464	3.53643	9.69375
240	451.978	2835.52	3215.13	2926.63	3106.92	13.3877	3.2132	9.57143
280	525.389	2863.43	3291.52	2955.3	3175.16	14.9502	3.20827	10.8863
320	626.744	2775.11	3235.78	2894.63	3074.89	16.5999	4.30693	10.8024
360	715.389	2913.18	3374.4	3026.26	3317.4	15.8323	3.8816	13.8756
400	821.822	3150.07	3603.43	3304.29	3573.03	14.3923	4.89584	13.4272
440	894.922	2983.16	3496.91	3112.83	3325.01	17.2219	4.347	11.4595
480	973.078	3060.02	3678.7	3182.17	3378.76	20.2181	3.99162	10.416
520	1061.74	3053.21	3846.96	3194.8	3443.17	25.997	4.63738	12.772

Table 4. Results of our main experiments on mode 2 random chordal graphs, evaluating heuristics for the interval coloring problem.

$\|V\|$	$\|E\|$	OPT	Best Fit	First Fit	Partition	$\frac{(BFI-OPT)}{OPT}$	$\frac{(FFI-OPT)}{OPT}$	$\frac{(GPI-OPT)}{OPT}$
40	109.689	3094.16	3138.77	3216.93	3377.4	1.44179	3.96805	9.15418
80	302.489	3927.78	4074.87	4175.86	4441.82	3.74484	6.31598	13.0874
120	524.8	4459.87	4623.33	4811.37	5110.5	3.66528	7.8814	14.5886
160	799.044	4891.27	5002.19	5306.62	5711.9	2.26776	8.49178	16.7775
200	1052.07	5290.47	5473.74	5793.81	6204.62	3.4643	9.51418	17.2793
240	1363.43	5390.86	5600.83	5896.01	6317.99	3.89507	9.3706	17.1983
280	1655.84	5749.18	5988.46	6283.7	6728.99	4.16195	9.29737	17.0426
320	1953.24	5776.97	6017.8	6364.53	6816.09	4.16885	10.1709	17.9873
360	2261.96	5899.47	6249.91	6552.94	7039.04	5.94027	11.0769	19.3166
400	2639.39	5987.34	6255.47	6616.7	7155.96	4.47815	10.5114	19.518
440	2956.79	6161.91	6446.57	6807.86	7327.6	4.6196	10.4829	18.9177
480	3243.42	6236.88	6569.51	6833.61	7423.2	5.33333	9.56782	19.0211
520	3645.54	6296.03	6573.81	6980.31	7555.84	4.41195	10.8684	20.0096

	MODE 1			MODE 2		
	BFM	FFM	GPM	BFM	FFM	GPM
Equals OPT	1216	2965	0	4791	1515	2
Equals χ	4950	3393	1	4950	1675	6
% Deviation	10.70	1.93	58.66	0.20	4.47	39.43
	Random Weights					
Equals LB	53	71	0	38	57	0
Equals χ	4950	2666	3	4950	441	7
% Deviation	24.64	17.43	78.67	36.20	25.88	59.13

Fig. 5. This table shows aggregate performance over all 4950 runs of the three heuristics for max-coloring, separately for mode 1 and mode 2 graphs. The first three rows correspond to the experiments where the value of OPT is known. The next three rows correspond to the runs where the weights were assigned randomly. In this case, the algorithms are compared against the weight of the maximum weight clique (LB). The row "Equals OPT(LB)" lists the number of times each heuristic produces a coloring with weight equal to OPT(LB), 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(LB), averaged over the 4950 runs.

2. The graph partitioning heuristic is not competitive at all, relative to first-fit or best-fit, in any of the cases, despite the $O(\log n)$-factor approximation guarantee it provides.
3. Between the first-fit heuristic and the best-fit in reverse perfect elimination order heuristic, first-fit seems to do better in an overall sense. However, they exhibit opposite trends in going from mode 1 to mode 2 graphs. Specifically, first-fit's performance worsens with the percentage deviation changing as $1.93 \rightarrow 4.47$ for max-coloring, while best-fit's performance improves with the percentage deviation going from $10.70 \rightarrow 0.20$. Figure 5 and the graphs in Figure 7, show the performance of the three algorithms on mode 1 and mode 2 graphs. We see the same trend for interval coloring as well. The performance of first-fit worsens as we go from mode 1 to mode 2 graphs, while best-fit's performance improves. Figure 6 and the graphs in Figure 8 show the deviations from OPT for interval coloring. In order to verify this trend, we tested the algorithms on mode 1 and mode 2 graphs, with randomly assigned vertex weights. First-fit's performance continues to show this trend of deteriorating performance in going from mode 1 to mode 2 graphs for both max-coloring and interval coloring. However the performance of best-fit is quite different. It's performance on max-coloring worsens as we go from mode 1 to mode 2 graphs, while it improves slightly for interval coloring.
4. Best-fit heuristic seems to be at a disadvantage because it is constrained to use as many colors as the chromatic number. First-fit uses more colors than the chromatic number a fair number of times. Examining figure 5 we note

	MODE 1			MODE 2		
	BFI	FFI	GPI	BFI	FFI	GPI
Equals OPT	2878	3368	2044	2459	1353	282
% Deviation	9.51	2.29	7.89	6.70	9.52	17.45
Random Weights						
Equals LB	1755	1568	939	467	211	110
% Deviation	26.87	10.99	16.91	25.21	24.52	30.01

Fig. 6. This table shows aggregate performance over all 4950 runs of the three heuristics for interval coloring, separately for mode 1 and mode 2 graphs. The first three rows correspond to the case where the value of OPT is known. The next three rows correspond to the case where the weights were assigned randomly. In the latter case, the performance of the algorithms are compared with the weight of the maximum weight clique (LB). The row "Equals OPT(LB)" lists the number of times each heuristic produces a coloring with weight equal to OPT(LB) and the row "% Deviation" lists the percentage deviation of the weight of the solution produced from OPT(LB), averaged over the 4950 runs.

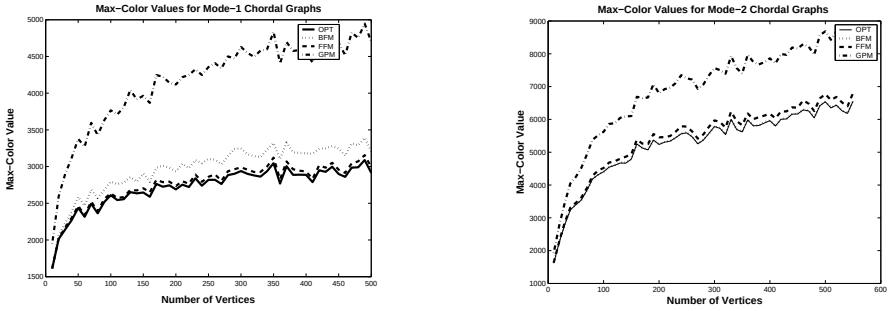

Fig. 7. Graphs showing values for max-coloring mode 1 and mode 2 chordal graphs. The x-axis corresponds to the number of vertices in the graph, and the y-axis corresponds to the max-color value. The solid line shows the value of OPT for the different sizes of the graph, and the dashed line corresponds to the value of the coloring produced by first-fit; the dotted line, the performance of best-fit; and the dashed-dotted line, the performance of the graph partitioning heuristic.

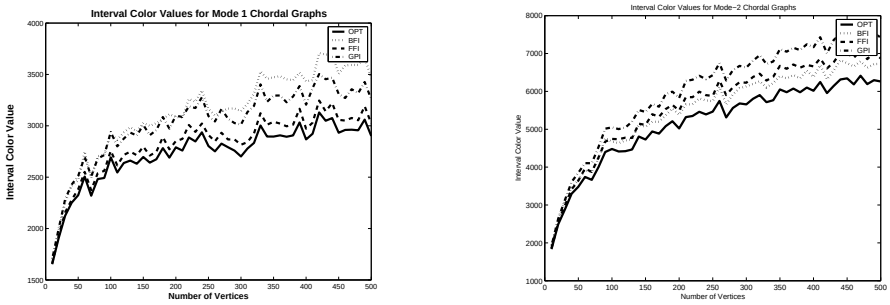

Fig. 8. Graph showing values for interval coloring mode 1 and mode 2 chordal graphs. The x-axis corresponds to the number of nodes in the graph, and the y-axis corresponds to the interval color value. The solid line corresponds to OPT; the dotted line, to the performance of best-fit; the dashed line to the performance of first-fit; and the dashed-dotted line, to that of the graph partitioning heuristic.

that for max-coloring, first-fit uses more colors than OPT about 31.45% and 66.16% of the time for mode 1 and mode 2 graphs respectively.

4 Conclusion

Our goal was to evaluate the performance of three simple heuristics for the max-coloring problem and for the interval coloring problem. These heuristics were first-fit, best-fit, and a heuristic based on graph partitioning. First-fit outperformed the other algorithms in general, and our recommendation is that this be the default heuristic for both problems. Despite the logarithmic approximation guarantee it provides, the heuristic based on graph partitioning is not competitive in comparison to first-fit or best-fit. Best-fit seems to perform better on graphs that are more "irregular" and offers first-fit competition for such graphs. We have also experimented with other classes of chordal graphs such as trees and sets of disjoint cliques. Results from these experiments are available at `http://www.cs.uiowa.edu/~rraman/chordalGraphExperiments.html`

References

1. A.L. Buchsbaum, H. Karloff, C. Kenyon, N. Reingold, and M. Thorup. OPT versus LOAD in dynamic storage allocation. In *Proceedings of the 35th Annual ACM Symposium on Theory of Computing (STOC)*, 2003.
2. M. Chrobak and M. Ślusarek. On some packing problems related to dynamic storage allocation. *Informatique théorique et Applications/Theoretical Informatics and Applications*, 22(4):487–499, 1988.
3. G. Confessore, P. Dell'Olmo, and S. Giordani. An approximation result for the interval coloring problem on claw-free chordal graphs. *Discrete Applied Mathematics*, 120:71–88, 2002.
4. J. Fabri. Automatic storage optimization. *ACM SIGPLAN Notices: Proceedings of the ACM SIGPLAN '79 on Compiler Construction*, 14(8):83–91, 1979.
5. M.R. Garey and D.S. Johnson. *Computers and Intractability: A Guide to the theory of NP-completeness.* W.H. Freeman and Company, San Fransisco, 1979.
6. J. Gergov. Approximation algorithms for dynamic storage allocation. In *Proceedings of the 4th European Symposium on Algorithms: Lecture Notes in Computer Science 1136*, pages 52–61, 1996.
7. J. Gergov. Algorithms for compile-time memory optimization. In *Proceedings of the 10th ACM-SIAM Symposium on Discrete Algorithms*, pages S907–S908, 1999.
8. R. Govindarajan and S. Rengarajan. 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*, 1996.
9. S. Irani. Coloring inductive graphs on-line. *Algorithmica*, 11(1):53–72, 1994.
10. H.A. Kierstead. The linearity of first-fit coloring of interval graphs. *SIAM J. Discrete Math*, 1:526–530, 1988.
11. H.A. Kierstead. A polynomial time approximation algorithm for dynamic storage allocation. *Discrete Mathematics*, 88:231–237, 1991.
12. H.A. Kierstead and W.T. Trotter. An extremal problem in recursive combinatorics. *Congressus Numerantium*, 33:143–153, 1981.
13. S.V. Pemmaraju, R. Raman, and K. Varadarajan. Buffer minimization using max-coloring. To appear in the Proceedings of The ACM-SIAM Symposium on Discrete Algorithms (SODA), 2004.
14. T. Rus and S.V. Pemmaraju. Using graph coloring in an algebraic compiler. *Acta Informatica*, 34(3):191–209, 1997.
15. R.E. Tarjan and M. Yannakakis. 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, 1984.

Appendix

Max-Coloring Mode 1 graphs with random weights

$\|V\|$	$\|E\|$	LB	Best Fit	First Fit	Partition	$\frac{(BF-OPT)}{OPT}$	$\frac{(FF-OPT)}{OPT}$	$\frac{(GPM-OPT)}{OPT}$
40	64.7222	2793.91	3346.9	3148.41	4043.36	19.7926	12.6883	44.7203
80	144.656	3057.64	3759.9	3499.66	4668.01	22.9672	14.4559	52.6669
120	217.022	3177.24	3946.64	3668.24	5015.84	24.216	15.4536	57.8678
160	310.744	3263.64	4123.96	3863.77	5306.64	26.3604	18.3881	62.5987
200	369.5	3436.68	4236.58	4049.02	5544.8	23.2754	17.8179	61.3419
240	465.822	3555.23	4466.49	4171.19	5732.43	25.6314	17.3253	61.2393
280	543.056	3670.51	4664.58	4319.58	5981.14	27.0825	17.6833	62.9513
320	616.833	3683.39	4599.39	4257.07	5971.09	24.8684	15.5747	62.1086
360	725.544	3675.27	4721.42	4376.39	6133.7	28.4648	19.0768	66.8913
400	799.322	3718.68	4700.19	4412.16	6158.34	26.3941	18.6485	65.6058
440	853.956	3766.96	4733.96	4428.67	6248.27	25.6706	17.5662	65.8705
480	953.733	3708.09	4750.59	4444.38	6195.4	28.1142	19.8563	67.078
520	1051.66	3788.9	4818.58	4490.54	6385.32	27.1762	18.5184	68.5271

Max-Coloring Mode 2 graphs with random weights

Table 5. Results of our main experiments on mode 2 random chordal graphs, with random weights evaluating heuristics for the max-coloring problem.

$\|V\|$	$\|E\|$	LB	Best Fit	First Fit	Partition	$\frac{(BFI-OPT)}{OPT}$	$\frac{(FFI-OPT)}{OPT}$	$\frac{(GPI-OPT)}{OPT}$
40	108.989	3579.93	4492.93	4144.81	4864.37	25.5033	15.779	35.8787
80	300.522	4545.92	5842.42	5302.06	6248.58	28.5201	16.6332	37.4546
120	534.711	5007.94	6674.97	6027.22	7103.28	33.2876	20.3532	41.8402
160	799.078	5477.86	7339.93	6594.4	7700.49	33.9928	20.3829	40.5749
200	1032.77	5613.7	7683.96	6792.93	8030.8	36.8786	21.0063	43.0572
240	1351.2	5877.72	8096.71	7077.89	8455.37	37.7525	20.4189	43.8545
280	1670.63	6349.47	8686.23	7656.33	9033.13	36.8026	20.5823	42.266
320	1977.26	6538.11	8888.31	7837.77	9316.49	35.9462	19.8782	42.4951
360	2259.06	6500.41	9195.9	7919.62	9315.26	41.4664	21.8326	43.3026
400	2579.76	6623.37	9270.36	8096.2	9643.78	39.9644	22.2369	45.6024
440	2991.97	6919.62	9724.36	8390.56	9996.6	40.533	21.2574	44.4674
480	3315.63	7062.68	9891.43	8462.93	10052.4	40.0522	19.8261	42.3314
520	3615.3	7151.23	10029.5	8588.54	10183.7	40.2487	20.0988	42.405

Interval Coloring Mode 1 graphs with random weights

Table 6. Results of our main experiments on mode 1 random chordal graphs, with random weights evaluating heuristics for the interval coloring problem.

$\|V\|$	$\|E\|$	LB	Best Fit	First Fit	Partition	$\frac{(BFI-OPT)}{OPT}$	$\frac{(FFI-OPT)}{OPT}$	$\frac{(GPI-OPT)}{OPT}$
40	65.4667	2793.79	3182.16	3015.37	3139.42	13.9011	7.93109	12.3715
80	138.911	3025.34	3730.73	3361.86	3572.19	23.316	11.1231	18.0754
120	221.489	3246.92	4213.96	3573.64	3803.58	29.7831	10.0625	17.1441
160	304.589	3349.5	4345.77	3773.23	4031.22	29.7437	12.6506	20.353
200	388.922	3487.59	4702.46	3960.63	4250.49	34.834	13.5637	21.8747
240	465.622	3564.61	4774.09	3985.03	4317.4	33.9301	11.7943	21.1184
280	547.156	3621.37	4935.97	4079.52	4346.26	36.3012	12.6515	20.017
320	646.844	3632.78	5053.3	4158.73	4420.82	39.1029	14.4781	21.6926
360	708.389	3636.84	5215.73	4140.89	4413.28	43.4137	13.8594	21.3491
400	800.833	3790.56	5389	4357.7	4544.91	42.1691	14.962	19.9009
440	890.633	3751.67	5564.04	4290.47	4549.29	48.3086	14.3616	21.2605
480	972.989	3770.38	5594.34	4365.38	4627.62	48.3762	15.7809	22.7363
520	1034.57	3785.07	5645.89	4345.94	4671.27	49.1622	14.8182	23.4131

Interval Coloring Mode 2 graphs with random weights

Table 7. Results of our main experiments on mode 2 random chordal graphs with random weights, evaluating heuristics for the interval coloring problem.

$\|V\|$	$\|E\|$	LB	Best Fit	First Fit	Partition	$\frac{(BFI-OPT)}{OPT}$	$\frac{(FFI-OPT)}{OPT}$	$\frac{(GPI-OPT)}{OPT}$
40	109.3	3619.4	3960.23	4130.19	4353.67	9.41685	14.1125	20.287
80	305.778	4599.7	5216.71	5446.9	5729.69	13.4142	18.4186	24.5666
120	534.244	5080.24	5892.5	6148.76	6414.76	15.9885	21.0327	26.2686
160	791.522	5379.14	6460.83	6590.64	6915.19	20.1089	22.5222	28.5556
200	1076.37	5854.59	6830.21	7224.81	7488.83	16.6642	23.4042	27.9139
240	1338.13	5969.88	7236.98	7329.67	7760.87	21.2249	22.7775	30.0004
280	1685.17	6280.87	7541.18	7765.64	8109.21	20.0659	23.6397	29.1097
320	1911.62	6281.07	7692.9	7842.99	8220.39	22.4776	24.8671	30.8757
360	2308.61	6597.4	8010.04	8071.09	8564.29	21.4121	22.3374	29.8131
400	2584.28	6800	8210.2	8375.16	8740.73	20.7382	23.1641	28.5402
440	2958.31	6940.89	8431.81	8543.13	9072.14	21.4803	23.0841	30.7058
480	3251.54	7010.97	8681.73	8574.38	9276.71	23.8308	22.2995	32.3171
520	3625.52	7366.5	9047.21	9139.46	9543.09	22.8156	24.0678	29.5471