Mutual Exclusion Scheduling

Brenda S. Baker
Edward G. Coffman, Jr.

AT&T Bell Laboratories
600 Mountain Avenue
Murray Hill, NJ 07974
bsb@research.att.com
egc@research.att.com

March 5, 1996

Abstract

Mutual exclusion scheduling is the problem of scheduling unit-time tasks non-preemptively on m processors subject to constraints represented by a graph G, such that tasks represented by adjacent vertices in G must run in disjoint time intervals. This problem arises in load-balancing the parallel solution of partial differential equations by domain decomposition. Minimizing the completion time is NP-hard even if either the number of processors or the completion time is fixed but greater than two. However, polynomial time is sufficient to produce optimal schedules for forests, and simple heuristics perform well on certain classes of graphs. For graphs derived from the two-dimensional domain decomposition problem, heuristics yield solutions within $4c - 7$ time units of optimal, where c is the maximal number of regions that touch each other at a single point in the domain decomposition; these solutions are within a constant factor of optimal.

1. Introduction

This paper studies the problem of scheduling tasks constrained by a *mutual exclusion* graph G in which each vertex represents a task requiring one unit of running time. The tasks must be scheduled nonpreemptively on $m \geq 2$ identical processors so that tasks represented by adjacent vertices in G run in disjoint (mutually exclusive) time intervals. The problem, which we call MUTUAL EXCLUSION SCHEDULING, is to minimize the makespan, or completion time, of the schedule, subject to the mutual exclusion constraints.

Mutual exclusion scheduling arises in load-balancing the parallel solution of partial differential equations (pde's) by domain decomposition. The two or three dimensional domain for the pde's is decomposed into regions, each region corresponding to a subcomputation. The decomposition may be chosen so that the predicted subcomputation times are approximately equal. The subcomputations are to be scheduled on m processors so that subcomputations corresponding to regions that touch even at a single point are not performed simultaneously, and so that the makespan is minimized. Such a system for the parallel solution of pde's is being developed by Bjørstad, Coughran, and Grosse [5]. In their implementation, they require a static schedule that will be reused for multiple iterations to avoid having communication costs dominate the run time. The domain decomposition scheduling problem is transformed into MUTUAL EXCLUSION SCHEDULING by extracting a graph G from the domain decomposition, such that each region of the domain decomposition is represented by a single vertex in the graph, and two vertices in the graph are adjacent if and only if the corresponding regions touch at one or more points.

An example of an application area is semiconductor device simulation, in which the pde's apply to irregular physical structures of different sizes and composed of different materials. The regions obtained through decomposition may be irregular in shape and of different sizes even though computation times are expected to be approximately equal. Different regions may touch different numbers of other regions; in particular, a region corresponding to the underlying substrate may touch many other regions, while interior regions may not. Thus, the graph for the resulting MUTUAL EXCLUSION SCHEDULING problem has an irregular structure.

MUTUAL EXCLUSION SCHEDULING without the processor constraint ($m \geq n$ for all instances) becomes a scheduling (timetabling) problem studied nearly 30 years ago by Welsh and Powell [12]. Note that the decision version of this unconstrained problem is equivalent to CHROMATIC NUMBER (see, e.g., [8, p. 191]). We will return shortly to the obvious connections between coloring algorithms and MUTUAL EXCLUSION SCHEDULING.

We can decide complexity issues for MUTUAL EXCLUSION SCHEDULING by examining the complexity of its decision version; in terms of graphs, the latter is called BOUNDED INDEPENDENT SETS: *For given m and t, determine whether G can be partitioned into at most t independent sets with at most m vertices in each.* Bodlaender and Jansen [6] introduced this problem, but as the decision version of a complementary scheduling problem. Their initial interest was in COMPATIBILITY SCHEDULING which has the same instance and makespan objective function as MUTUAL EXCLUSION SCHEDULING but which places a different meaning on adjacency in G; if two tasks are adjacent in G then they ca not be run on the same processor, i.e., they are incompatible. Thus, in mutual exclusion schedules an independent set is comprised of the tasks running in a time unit, whereas in a compatibility schedule, it is comprised of the tasks running on a processor.

Lonc's results [10] showed that, for split graphs, BOUNDED INDEPENDENT SETS can be solved in polynomial time. However, Bodlaender and Jansen [6] established that the problem was NP-complete when G is restricted to cographs, bipartite graphs, or interval graphs. They also proved the following results: If either t or m is a fixed constant, then BOUNDED INDEPENDENT SETS is in P for cographs; if t is a fixed constant, then the problem is in P for interval graphs, and if m is a fixed constant, then it is in P for bipartite graphs. BOUNDED INDEPENDENT SETS remains NP-complete for bipartite graphs and any fixed $t \geq 3$, and for interval graphs and any fixed $m \geq 4$. The problem for interval graphs and $m = 3$ is open. Finally, the problem for co-interval graphs can be solved in time linear in the number of tasks.

In the next section, we extend these complexity results by considering general graphs with fixed m or t and graphs restricted to be forests. We show that BOUNDED INDEPENDENT SETS is in P for $m = 2$ or $t = 2$, but is NP-complete for fixed $m \geq 3$ and (from the above bipartite graph result) for fixed $t \geq 3$. We prove that if G is restricted to forests, then an optimal mutual exclusion schedule can be found in $O(n + m^2 \log m)$ time, where n is the number of tasks and m is the number of processors. The former results are relatively easy, but the result for forests requires some effort. As part of the forest result, we find that $O(n)$ time suffices to find optimal mutual exclusion schedules for trees.

For arbitrary graphs, it is natural to seek polynomial-time approximation algorithms. We will verify that, unfortunately, there is a $\delta > 0$ such that there does not exist a polynomial time approximation algorithm A that achieves $A(G,m,t)/OPT(G,m,t) < n^{\delta}$ for a graph G with n vertices unless $P = NP$.

The independent sets obtained by coloring algorithms can be used as the basis of approximate solutions; indeed, this approach has been taken in the literature on domain decomposition [5]. A simple algorithm of this type colors the graph with at most $d+1$ colors, where d is the maximum degree of the graph. Using a greedy algorithm, each successive vertex is given a color different from that of any neighbor already colored. This algorithm runs in time linear in the number of edges in G. The independent set corresponding to a color class of r vertices can be trivially scheduled on m processors in makespan $\lceil r/m \rceil$. A mutual exclusion schedule for the whole graph is obtained by taking the schedules for the various colors in succession. The makespan of the resulting schedule is at most $\lfloor n/m \rfloor + d + 1$, where n is the number of vertices, and d is the maximum vertex degree. As an example, for planar graphs, which are

4-colorable in polynomial time [2], coloring-based scheduling comes within 4 of optimal. The literature on coloring algorithms for general graphs also includes more computation-intensive polynomial-time algorithms such as the Berger-Rompel algorithm [4], which colors any k-colorable graph using $O((n/log)^{(1-1/(k-1))})$ colors, and polynomial-time algorithms that color random k-colorable graphs optimally with high probability. (For a discussion of the latter approach, see [1].)

In the final part of Section 2, we describe a simple greedy heuristic, called Greedy Mutual Exclusion (GME), which improves on the performance guarantee of the coloring-based scheduling heuristic. With a given ordering of the vertices, GME schedules vertices one at a time into the earliest time unit such that mutual exclusion constraints are met and at most m vertices per time unit are scheduled. GME runs in time linear in the number of edges. If the vertex ordering is by decreasing degree, then for m processors and any graph G with n vertices, GME generates a schedule with makespan at most $OPT + deg(v_k)$, with $k = \lceil n/m \rceil + 1$, where v_k is the vertex of kth largest degree, $deg(v_k)$ is its degree, and where OPT is the makespan of an optimal schedule. Note that the additive constant of the bound has been reduced from one plus the maximum vertex degree for greedy coloring-based scheduling to the kth largest degree for GME.

For specific G, the performance of GME can be much better. For example, if G is a forest, then time linear in the number of vertices is sufficient to find a vertex ordering under which GME achieves a makespan within one of optimal; the algorithm is much simpler than the complicated optimization algorithm given in Section 2.1. Outerplanar graphs give another example. (Recall that these are planar graphs that can be laid out so that every vertex is on an exterior face.) For an outerplanar graph, time linear in the number of vertices is sufficient to order the vertices so that GME obtains a mutual exclusion schedule having a makespan within 2 of optimal.

Section 3 explores mutual exclusion scheduling for the two-dimensional domain decomposition problem defined by Coughran and Grosse. The mutual exclusion graph created by a two-dimensional domain decomposition is generally nonplanar. However, it is a natural dual to the planar decomposition, although not the standard graph-theory definition of dual, which would yield a planar graph. For such a mutual exclusion graph, if the maximum number of regions touching at a single point is $c > 2$, and there are n regions, our algorithm produces an m-processor mutual exclusion schedule with makespan at most $\lceil n/m \rceil + 4c - 7 \leq OPT + 4c - 7$.

Since $OPT \geq c$, this bound is within a constant times optimal.

To compare this result with the $\lfloor n/m \rfloor + d + 1$ bound of the greedy coloring-based scheduling algorithm, recall first that the maximum degree d of G represents the total number of regions that touch any single region at a point or edge. Thus, if some interior region has e edges, and each edge endpoint is on the boundary of $c > 2$ regions, then $d \geq e(c-2)$ and the worst-case bound for the makespan produced by the coloring algorithm is at least $\lfloor n/m \rfloor + e(c-2) + 1$, whereas the worst-case bound for our algorithm is at most $\lceil n/m \rceil + 4(c-2) + 1$.

The following notational conventions are observed throughout the remainder of the paper. We reserve m for the number of processors, and $G = (V, E)$ for the mutual exclusion graph, where V and E are the sets of vertices and edges in G. For any mutual exclusion scheduling algorithm A and graph G, let $A(G)$ be the makespan produced by A for G. For a vertex v in a graph, let $deg(v)$ be the degree of v. We assume that graphs are represented by giving a list of edges for each vertex.

2. Results

Section 2.1 focuses on new complexity results. Section 2.2 concludes with performance bounds on heuristic methods.

2.1 Complexity Reducibility from coloring problems suffices to prove the NP-completeness of BOUNDED INDEPENDENT SETS. Indeed, since 3-COLORABILITY is NP-complete even for planar graphs [8], the result holds with G restricted to planar graphs. The following two theorems give a more detailed picture.

Theorem 1. *For any graph G, an optimal 2-processor mutual exclusion schedule can be found in polynomial time. However, for any $m \geq 3$, the problem of finding an optimal schedule is NP-complete.*

Proof. Construct the complement of G, in which two vertices are adjacent if and only if they can be scheduled simultaneously. Observe that, for $m = 2$, a maximal matching in the complement graph yields an optimal mutual exclusion schedule. Since a maximal matching can be found in polynomial time, the theorem follows for $m = 2$.

It is easy to see that, with m fixed at 3, the complexity of MUTUAL EXCLUSION SCHEDULING is the same as that of the NP-complete PARTITION INTO TRIANGLES [8] in the complement graph. Then we can show NP-hardness for all $m \geq 3$ by induction, with $m = 3$ as the basis. For, the m-processor problem is polynomial-time reducible to the $(m+1)$-processor problem. To see this, simply add a clique of t vertices as a new component to the graph G in an instance for m processors. The new graph has an $(m+1)$-processor schedule with makespan t if and only if G has an m-processor schedule with makespan t. Obviously, MUTUAL EXCLUSION SCHEDULING is in NP for all m. □

Theorem 2. *The problem of determining for an arbitrary graph $G = (V, E)$ and a positive integer m whether there is a mutual exclusion schedule with makespan $t = 2$ is solvable in $O(|E| + m^2 \log m)$ time. However, the problem is NP-complete for any fixed $t \geq 3$.*

Proof. Suppose $t = 2$, and assume G is bipartite and $n \leq 2m$, since otherwise a schedule of makespan 2 is not possible. The 2 color classes of a connected bipartite graph are unique, so if G is connected, then makespan 2 is achievable if and only if each color class has at most m tasks. If G is not connected, then the addition of $2m - n$ independent tasks to G can not change the decision, so we now assume $n = 2m$ for simplicity. We use a standard dynamic programming approach, an extension of the pseudo-polynomial time algorithm for PARTITION given in [8, pp. 90–91].

Let $G_1, \ldots, G_\tau$, $\tau > 1$, be the connected components of G, and let j_i, k_i denote the color-class sizes of G_i, $1 \leq i \leq \tau$. Define the Boolean function $f(i, j, k)$ to have the value TRUE if and only if there is a coloring of $G_1, \ldots, G_i$ such that the two color classes of this subgraph of G have sizes j and k. In terms of f, the answer YES to our decision problem applies if and only if $f(\tau, m, m) = \mathit{TRUE}$. The function f can be tabulated by evaluating the recurrence

$$\begin{aligned} f(0, j, k) &= \mathit{TRUE} \text{ if and only if } j = k = 0 \\ f(i, j, k) &= f(i-1, j-j_i, k-k_i) \vee f(i-1, j-k_i, k-j_i), \quad 1 \leq i \leq \tau \ . \end{aligned}$$

For any i, there are at most $m + 1$ nonzero table entries $f(i, j, k)$ with $j, k \leq m$ It is easy to verify that, if a balanced tree is used to manage the nonzero elements of the table, the evaluation of $f(\tau, m, m)$ can be done in $O(m^2 \log m)$ time. Together with the fact that the bipartite property of G can be determined in time linear in the number of edges, this result proves the theorem for $t = 2$.

It remains to observe from the results in [6] that NP-completeness holds for fixed $t \geq 3$ even when G is restricted to bipartite graphs. □

The next theorem shows that forests can be scheduled optimally in polynomial time.

Theorem 3. *For a forest G of n tasks, an optimal mutual exclusion schedule on m processors can be computed in $O(n+m^2 \log m)$ time. If the forest is just a single tree, then the computation can be done in only $O(n)$ time.*

Proof. We begin with two claims disposing of easily proved cases.

Claim 1. *The theorem holds if $n \leq 2m$.*

Proof. Since a forest is bipartite, this claim is an easy consequence of Theorem 2. In particular, for the case $n \leq 2m$, it is routine to convert the dynamic programming algorithm in the proof of Theorem 2 to an optimal m-processor mutual exclusion scheduling algorithm requiring $O(m^2 \log m)$ time, or $O(m)$ time in the case of trees. The details are left to the interested reader. □

The remainder of the argument assumes a two-coloring of G, which is always possible since G is a forest and hence bipartite. The coloring uses the colors red and blue, with $r = r(G)$ and $b = b(G)$ denoting the respective numbers of reds and blues, i.e., red and blue tasks. Without loss of generality, *we assume a coloring such that $r \leq b$ and all isolated tasks are blue.* Note that the coloring of G can be done in $O(n)$ time.

Schedules produced by the following algorithm will be called A_1-schedules.

Algorithm A_1

1. Schedule the reds, if any, in time units $1, \ldots, \lceil \frac{r}{m} \rceil$, with m reds in each of the first $\lfloor \frac{r}{m} \rfloor$ time units, and with the reds in time unit $\lceil \frac{r}{m} \rceil$ being of the smallest degree.

2. Then, schedule the blues in any order, placing each such blue B into the earliest unfilled time unit having no red adjacent to B.

Claim 2. *The theorem holds if $r \bmod m = 0$ or if $r > m$ and hence $b > m$.*

Proof. We show that, in these two cases, the A_1-schedule has the minimum makespan $\lceil \frac{n}{m} \rceil$. This is trivial to see if $r \bmod m = 0$, so assume that $m < r \leq b$ and $r \bmod m \geq 1$.

Let $d_1 \leq \cdots \leq d_p$, $p = r \bmod m \geq 1$, denote the degrees of the $p < m$ reds scheduled in time unit $\lceil \frac{r}{m} \rceil$.

Case 1. $d_p \leq 1$. We have $\sum_{i=1}^{p} d_i \leq p$, so at most p blues are adjacent to the reds in time unit $\lceil \frac{r}{m} \rceil$. Since $b > m$, there are at least $m - p$ blues independent of the reds in time unit $\lceil \frac{r}{m} \rceil$. Then Step 2 fills time unit $\lceil \frac{r}{m} \rceil$ and the resulting schedule has the minimal makespan $\lceil \frac{n}{m} \rceil$.

Case 2. $d_p \geq 2$. Since $r > m$, there are m reds scheduled in time unit $\lfloor \frac{r}{m} \rfloor = \lceil \frac{r}{m} \rceil - 1$. Let G' be the subforest induced by the reds in time units $\lfloor \frac{r}{m} \rfloor, \lceil \frac{r}{m} \rceil$ (a blue of G is in G' if and only if it is adjacent to one of these reds). The number of edges in G' is at least $md_p + \sum_{i=1}^{p} d_i$, since the reds in time unit $\lfloor \frac{r}{m} \rfloor$ have degrees at least that of any red scheduled in time unit $\lceil \frac{r}{m} \rceil$. The number of vertices in a forest exceeds the number of edges, so the number n' of tasks in G' is bounded by

$$n' = m + p + b' > md_p + \sum_{i=1}^{p} d_i \ ,$$

where b' is the number of blues in G'. Then

$$b' > m(d_p - 1) + \sum_{i=1}^{p} (d_i - 1) \ ,$$

so if b'' is the number of blues not adjacent to reds in time unit $\lceil \frac{r}{m} \rceil$, then

$$b'' \geq b' - \sum_{i=1}^{p} d_i > m(d_p - 1) - p \geq m - p \ ,$$

since $d_p \geq 2$. This implies that Step 2 always fills time unit $\lceil \frac{r}{m} \rceil$, which again implies that the A_1-schedule makespan is $\lceil \frac{n}{m} \rceil$.

Since the coloring requires $O(n)$ time, it remains only to observe that the steps of algorithm A_1 require $O(n)$ time. This is obvious for Step 2. It follows easily for Step 1 from the fact that, for any given k, the red with the k^{th} smallest degree can be found in $O(n)$ time. $\square$

By Claims 1 and 2 the remaining cases satisfy $n > 2m$, $1 \leq r < m < b$. The algorithm that covers these cases needs additional data structures computed from G. In a coloring of the forest G, the *leaf set* of the i^{th} red is the set of blue leaves adjacent to the i^{th} red, and is denoted S_i. We assume that the reds are indexed so that $|S_i| \geq \cdots \geq |S_r| \geq 0$. Define r_0 and r_1 as the respective numbers of reds with no blue leaves and exactly one blue leaf, and let $r_* = r - r_0 - r_1$ count the number of leaf sets with at least two blues. The reds counted by r_0, r_1 will be called 0-reds and 1-reds, respectively.

If L denotes a list of sets, then $|L|$ denotes the number of sets in L and $\#L$ denotes the number of elements in the union of the sets in L. Now partition $S_1, \ldots, S_{r_*}$ into two lists $L^{(1)}$, $L^{(2)}$ by the following greedy rule: Initialize $L^{(1)} = \varphi$, $L^{(2)} = \varphi$. For $j = 1, \ldots, r_*$: if $\#L^{(1)} + |L^{(2)}| \leq \#L^{(2)} + |L^{(1)}|$ then append S_j to $L^{(1)}$; otherwise, append S_j to $L^{(2)}$. The quantity $\#L^{(1)} + |L^{(2)}|$ can be interpreted as the total number of blues in leaf sets of $L^{(1)}$ plus the number of reds with leaf sets in $L^{(2)}$. The blues and reds thus counted can be scheduled in the same time unit, as can those counted by $\#L^{(2)} + |L^{(1)}|$. This fact is exploited in algorithm A_2 below.

Let $L^{(1)} = S_1^{(1)}, \ldots, S_s^{(1)}$, $L^{(2)} = S_1^{(2)}, \ldots S_{r_*-s}^{(2)}$ be the final greedy partition, and define

$$\Delta = [\#L^{(1)} + |L^{(2)}|] - [\#L^{(2)} + |L^{(1)}|] = \#L^{(1)} - \#L^{(2)} + r_* - 2s \ .$$

Since $r < m$ in the remaining cases, the time to compute the S_i, $1 \leq i \leq r$, and then order them in (n) time by placing them in n buckets representing numbers of leaves; and the time to construct $L^{(1)}, L^{(2)}$ is $O(n)$.

We conclude the proof by showing that the following algorithm is optimal for $n > 2m$, $1 \leq r < m < b$.

Algorithm A_2

Let t_i denote time unit i.

1. Schedule r_1 1-reds in t_1 and their adjacent blue leaves in t_2.
2. If $\Delta \neq 0$, put $\min\{r_0|\Delta|\}$ 0-reds in t_1 or t_2 according as $\Delta < 0$ or $\Delta > 0$, respectively. Then, if $r_0 > |\Delta|$, schedule the remaining 0-reds, $\left\lceil \frac{r_0 - |\Delta|}{2} \right\rceil$ in t_1 and $\left\lfloor \frac{r_0 - |\Delta|}{2} \right\rfloor$ in t_2.
3. Schedule in t_1 the $r_* - s$ reds with leaf sets $S_i^{(2)}$, and in t_2 the s reds with leaf sets $S_i^{(1)}$.
4. In t_1 schedule the blues, if any, in $S_1^{(1)}, \ldots, S_s^{(1)}$, taken in that order, until t_1 is filled or $S_s^{(1)}$ is exhausted, whichever occurs first. Repeat this procedure for t_2 with blues taken from $S_1^{(2)}, \ldots S_{r_*-s}^{(2)}$, in that order.
5. Schedule the remaining blues, if any, placing each such blue B into the earliest unfilled time unit not having a red adjacent to B.

For the optimality proof, define the following counts of tasks available for scheduling in t_1, t_2:

$$q^{(1)} = r^{(1)} + \sum_{1 \leq i \leq s} |S_i^{(1)}|, \quad q^{(2)} = r^{(2)} + \sum_{1 \leq i \leq r_* - s} |S_i^{(2)}| \ ,$$

where $r^{(i)}$ is the number of reds (including 0-reds and 1-reds) in t_i, $i = 1, 2$, at the conclusion of Step 3, and $r^{(1)} + r^{(2)} = r < m$. Note that Steps 1–4 schedule all reds in t_1, t_2 and a total of $\min\{m, q^{(i)}\}$ tasks in t_i, $i = 1, 2$. It is easy to see that, if $q^{(1)}$, $q^{(2)} \geq m$, then the makespan of the A_2-schedule is $\lceil \frac{n}{m} \rceil$. The same conclusion holds if one or both of these inequalities is violated, but Step 5 fills the gaps left by Step 4. Thus, in what follows, we assume that $q^{(i)} < m$ for $i = 1$ or 2, and at least one of t_1, t_2 is unfilled at the conclusion of the algorithm.

Case 1. $q^{(1)}$, $q^{(2)} \leq m$ with strict inequality for $q^{(1)}$ or $q^{(2)}$. Since at least one of t_1 and t_2 is unfilled in the final schedule, Step 4 must have scheduled all blue leaves in t_1, t_2 and Step 5 must have scheduled all isolated blues in t_1, t_2. Thus, the blues not scheduled in t_1, t_2 must be interior blues. Consider the subforest induced by the r reds and the b_{int} interior blues. Since there are at least 2 edges per interior blue, and since the number of tasks in the subforest must exceed the number of edges, we have $r + b_{int} \geq 2b_{int} + 1$ and hence

$$b_{int} \leq r - 1 \ . \tag{1}$$

Since $r < m$, a third time unit will accommodate the blues not scheduled in t_1, t_2. The final makespan will be 3, which is minimal, since $n > 2m$.

Case 2. $q^{(1)} > m$, $q^{(2)} < m$. For this case to hold, the leaf set $S_s^{(1)}$ must contribute at least one but not all of its blues to t_1; all other leaf sets must be scheduled entirely in t_1, t_2, by the greedy partition. Note also that, since t_2 is unfilled in the final schedule, all isolated blues must have been put in t_2 by Step 5. Thus, the number of blues scheduled after t_2 is at most the number b_{int} of interior blues plus the number $b_{rem} < |S_s^{(1)}|$ of blues remaining from $S_s^{(1)}$ after Step 4.

We first show that, if $s \geq 2$, then $b_{rem} \leq m - r + 1$. Together with (1), this implies that $b_{rem} + b_{int} \leq m$. Then only one additional time unit is required by Step 5, and the makespan is 3 as in Case 1.

To prove that $s \geq 2$ implies $b_{rem} \leq m - r + 1$, consider the iteration of the greedy rule when $S_s^{(1)}$ was appended to $L^{(1)}$. At that time, there must have been u complete leaf sets already assigned to $L^{(2)}$, with $1 \leq u \leq r_* - s$. (See Fig. 1.) By the greedy rule,

$$b_{rem} < |S_s^{(1)}| \leq |S_u^{(2)}| \ . \tag{2}$$

But $S_u^{(2)}$ is the smallest of $S_1^{(2)}, \ldots, S_u^{(2)}$, so its size is bounded by

$$|S_u^{(2)}| \leq \frac{1}{u} \sum_{1 \leq i \leq u} |S_i^{(2)}| \ . \tag{3}$$

To bound the sum in (3), we count the tasks scheduled in t_2 (see Fig. 1 for an illustration). In the (unsuccessful) attempt to balance the greedy partition, all r_0 0-reds were scheduled in t_2.

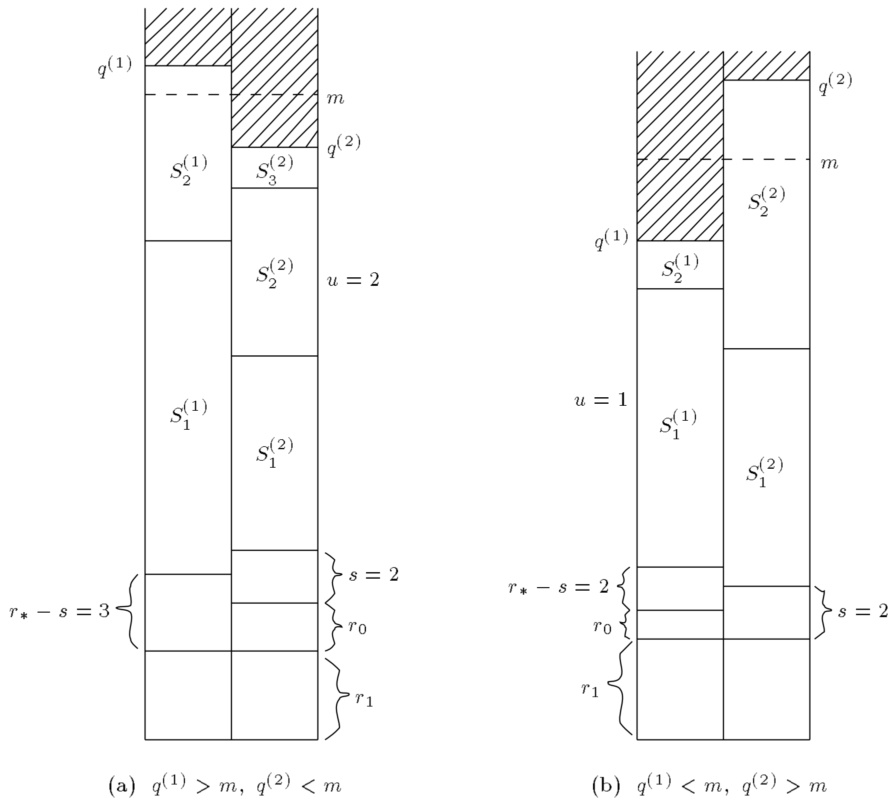

Figure 1: Examples, Cases 2, 3.

The r_1 blues from singleton leaf sets and the s reds with leaf sets $S_1^{(1)}, \ldots S_s^{(1)}$ are also in t_2. Counting the blues in the leaf sets $S_i^{(2)}$, we then have, since $q^{(2)} < m$,

$$r_0 + r_1 + s + \sum_{1 \leq i \leq r_* - s} |S_i^{(2)}| < m \ ,$$

and since $r = r_0 + r_1 + r_*$,

$$\sum_{1 \leq i \leq u} |S_i^{(2)}| \leq m - r + r_* - s - \sum_{u < i \leq r_* - s} |S_i^{(2)}| \ .$$

But $|S_i^{(2)}| > 1$, $1 \leq i \leq r_* - s$, so $\sum_{u < i \leq r_* - s} |S_i^{(2)}| > r_* - s - u$. Substituting, we get

$$\sum_{1 \leq i \leq u} |S_i^{(2)}| \leq m - r + u \ . \tag{4}$$

Then (2)–(4) give

$$b_{rem} < \frac{m - r + u}{u} \leq m - r + 1, \quad u \geq 1 \ ,$$

which is what we set out to prove.

Next, suppose that $s = 1$, i.e., only blues from $S_1^{(1)}$ are scheduled in t_1. Let R be the red with leaf set $S_1^{(1)}$. Since $q^{(2)} < m$, all leaf sets $S_2, \ldots S_{r_*}$ will be scheduled entirely in t_2 along with R. All 0-reds, all singleton leaf sets, all isolated blues, and all interior blues not adjacent to R are also scheduled with R in t_2. We claim that in *any* schedule of G, the time unit in which R is scheduled can have no more than the total of $q^{(2)}$ tasks listed above. To see this, consider the other possible tasks for t_2; these can only be reds in t_1. But all of these reds have leaf sets with at least one blue scheduled in t_2. Thus, if one of these reds were moved to t_2, at least one blue would have to be moved out of t_2. The claim follows. Since time units $1, 3, 4, \ldots$ contain a minimal makespan schedule for the subset of tasks run in these time units, the claim implies that the full schedule is optimal.

Case 3. $q^{(1)} < m$, $q^{(2)} > m$. Arguments similar to those in Case 2 apply. For this case, $S_{r_*-s}^{(2)}$ is the partially scheduled leaf set, with all other leaf sets scheduled entirely in t_1, t_2. Note that, by the greedy partition, the algorithm always puts S_1 in t_1, so we have $r_* - s \geq 2$, $s \geq 1$ in this case. Thus, we need only the argument for $s \geq 2$ in Case 2. As before, define $S_u^{(1)}$ as the last leaf set assigned to $L^{(1)}$ before $S_{r-s_*}^{(2)}$ was assigned to t_2. Clearly, $u \geq 1$, so the reasoning of Case 2 gives

$$b_{rem} < |S_{r_*-s}^{(2)}| \leq |S_u^{(1)}| \leq \frac{1}{u} \sum_{1 \leq i \leq u} |S_i^{(1)}| \ . \tag{5}$$

A count of the tasks in t_1 shows that (see Fig. 1)

$$r_0 + r_1 + r_* - s + \sum_{1 \leq i \leq s} |S_i^{(1)}| < m \ .$$

Then $|S_i^{(1)}| > 1$ and $r = r_0 + r_1 + r_*$ imply

$$\frac{1}{u} \sum_{1 \leq i \leq u} |S_i^{(1)}| \leq \frac{m - r + s - \sum_{u < i \leq s} |S_i^{(1)}|}{u} \leq \frac{m - r + u}{u} \leq m - r + 1, \quad u \geq 1 \ ,$$

which together with (5) gives $b_{rem} \leq m - r + 1$. Then by (1), $b_{rem} + b_{int} \leq m$ and the makespan is 3. The optimality of algorithm A_2 is proved.

Recall that the time required to compute the coloring, the ordered leaf sets, and the lists $L^{(1)}$, $L^{(2)}$ is $O(n)$. Algorithm A_2 clearly requires $O(n)$ time, so in conjunction with Claims 1 and 2 an optimal mutual exclusion schedule can be computed in $O(n + m^2 \log m)$ time for a forest, and in $O(n)$ time for a tree. $\square$

2.2 Heuristics We verify first that one can not expect approximation algorithms with good worst-case performance.

Proposition 1. *There exists a $\delta > 0$ such that there is no polynomial-time algorithm A for MUTUAL EXCLUSION SCHEDULING that achieves $A(G,m)/OPT(G,m) < n^{\delta}$ unless $P = NP$.*

Proof. As noted earlier, a graph G with n vertices has an n-processor schedule with makespan t if and only if G can be colored in t colors. But the proposition holds for the coloring problem, as shown by [11]. □

In spite of this negative result, a simple, linear-time greedy rule performs well for a large class of graphs, even in the worst case. Given a list L of vertices in G, the algorithm *Greedy Mutual Exclusion* (*GME*) schedules the vertices of G on m processors as follows. Call a time unit *full* if m tasks have been scheduled in that time unit. For each successive vertex v in L, GME schedules v in the earliest time unit that is not full and that contains no already-scheduled vertex adjacent to v. If L is in decreasing order of vertex degree, then the algorithm is called *Decreasing Greedy Mutual Exclusion* (*DGME*).

Lemma 1. *Let L be a list of the vertices in G, and define $s_L = \max_v p_L(v)$, where $p_L(v)$ is the number of vertices preceding v in L that are adjacent to v in G. Then $GME(G,m) \leq \lceil n/m + s_L(m-1)/m \rceil$.*

Proof. Let $t = \lceil n/m + s_L(m-1)/m \rceil$ and suppose $p_L(v) = r$. Since GRE schedules each vertex in the earliest time unit consistent with the mutual exclusion constraints, v is scheduled in time unit $t+1$ only if the first t time units are full, except for the at most r time units containing vertices adjacent to v. Therefore, at least $mt - r(m-1) \geq mt - s_L(m-1) \geq n$ vertices have already been scheduled, a contradiction. □

With this result, we can prove that GME is always near-optimal for forests and outerplanar graphs (see Figure 2).

Theorem 4. *If G is a forest, then its vertices can be listed in an order such that*

$$GME(G,m) \leq OPT(G,m) + 1$$

and if G is an outerplanar graph, then its vertices can be listed in an order such that

$$GME(G,m) \leq OPT(G,m) + 2$$

.

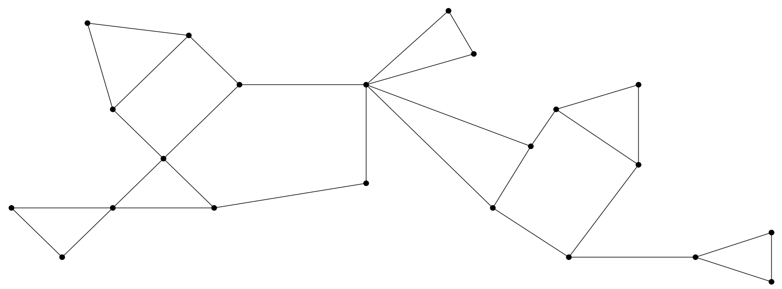

Figure 2: An outerplanar graph, i.e., a planar graph that can be laid out so that all vertices are on the exterior face.

Remark

Note that in Theorem 4 the lists of vertices can be constructed in time linear in the number of edges and GME runs in time linear in the number of edges. Thus, the GME heuristic is much simpler and faster than the optimization algorithm of the previous section, but the makespan is worse by only one time unit.

Proof. For a forest, a list is easily generated in which each vertex is adjacent to at most one previous vertex. The bound for forests then follows from Lemma 1.

Now suppose G is outerplanar. For convenience, the term 'bridge' refers to a face with one edge. From [3], the connectivity of the interior faces of an outerplanar graph G can be represented by an ordered tree in which each vertex represents an interior face of G, such that two tree vertices are adjacent if and only if the corresponding faces of G share a vertex or an edge. (In an outerplanar graph, two interior faces cannot share more than one edge and two vertices.) This tree can be constructed from G in time linear in the number of vertices. Pick any vertex of the tree as the root, and list the tree vertices according to a preorder traversal of the tree. Construct a list L of vertices of G as follows. For each face of G in order of the list of corresponding tree vertices, start at a vertex shared with the previous face (or with any vertex in the case of the first face), and list all as-yet-unlisted vertices of this face in counterclockwise order. Thus, a vertex is scheduled with the first face scheduled that contains it. Since each face is a cycle or a bridge, when a vertex is scheduled, it is adjacent to at most one vertex already scheduled, unless it is the last vertex scheduled in a cycle, in which case it is adjacent to two vertices already scheduled. Therefore, from Lemma 1, GME can schedule an outerplanar graph with makespan $2 + \lceil n/m \rceil$. □

For graphs not known to be of a special type such as forests or outerplanar graphs, for which a list can be created with a constant bound on how many neighbors can precede each vertex, putting the list of vertices in decreasing degree order yields a better guarantee than the bound of $OPT + d$ implied by Lemma 1, where d is the maximum degree. In particular, we have the following improved bounds. Let the vertices v_i of G be indexed so that $deg(v_1) \geq \ldots \geq deg(v_n)$.

Proposition 2. *With* $k = \lceil n/m \rceil$, *we have*

$$DGME(G, m) \leq \lceil n/m + deg(v_k)(m-1)/m \rceil \leq OPT(G, m) + deg(v_k)$$

A similar bound holds for $k = \max\{j \mid j \leq deg(v_j)\}$.

Proof. For any k, set $t_k = \lceil n/m + deg(v_k)(m-1)/m \rceil$. We claim that if $k \leq_k$, the schedule produced by DGME will be at most t_k. For the first k vertices are scheduled within t_k, and if vertex v_i, $i > k$, cannot be scheduled by t_k, there are at least $mt_k - (m-1)deg(v_i)$ $\geq mt_k - (m-1)deg(v_k) > n$ vertices already scheduled, a contradiction. To complete the proof, note that for the two values of k specified in the proposition, $k \leq t_k$. □

3. An Application to Domain Decomposition

This section investigates scheduling of mutual exclusion graphs resulting from the two-dimensional domain decomposition problem of Bjørstad et al. We define a domain decomposition to be an embedding of a 2-connected planar graph G in the plane.

The *edge dual* $D_E(G)$, or simply D_E if G is understood, is obtained from G as follows. For each interior face of G, $D_E(G)$ has a distinct vertex, and two vertices of $D_E(G)$ are connected by an edge if and only if the corresponding faces of the embedding of G share at least one edge. This definition is equivalent to the geometric dual in the terminology of [9, p. 113], except that we do not create a vertex for the exterior face. The restrictions on G imply that $D_E(G)$ has no self-loops or multiple edges. Furthermore, the planar embedding of G determines one of $D_E(G)$. A domain decomposition G and its edge dual are shown in Figure 3.

The *vertex dual* $D_V(G)$ of G, or simply D_V if G is understood, is obtained from the edge dual D_E by adding edges to create a clique within each face of the planar embedding of the edge dual. Figure 4 illustrates an edge dual and the corresponding vertex dual. Two vertices of $D_V(G)$ are connected by an edge if and only if the corresponding faces of the embedding of G share at least one vertex.

Theorem 5. *For any two-dimensional decomposition with* n *regions and any positive integer* m, *it is possible to find a mutual exclusion schedule for the vertex dual* D_V *with makespan*

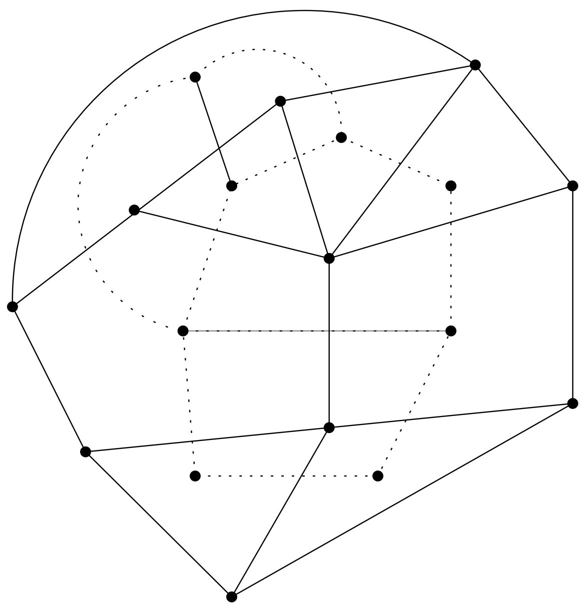

Figure 3: A planar decomposition (solid lines) and its edge dual (dotted lines).

$\lceil n/m \rceil + 4c - 7 \leq OPT + 4c - 7$, *where c is the size of the maximal number of regions touching at a single point in the planar decomposition and OPT is the makespan of an optimal schedule.*

Proof. To schedule a vertex dual, we make use of the underlying structure of the edge dual D_E. The following discussion refers to the edge dual. In D_E, label the vertices with level numbers as follows: label all vertices on the outermost face as level 1 vertices; and for each i, after removing all vertices of level i or less, label the vertices now on the outer face as level $i+1$ vertices. A level assignment is illustrated in Figure 5.

Note that all edges of D_E are between vertices at the same level or adjacent levels (i.e. level i and $i+1$ for some i). The same statement holds for edges of D_V since the additional edges are within faces of D_E.

Our goal is to construct a list containing all the odd vertices of G, such that in D_V each vertex is adjacent to at most $2c-4$ previous vertices in the list, and to construct a similar list for the even vertices. Then the two lists can be scheduled separately so that each schedule achieves the bound of Lemma 1. Concatenating the two schedules results in a schedule for the whole graph that achieves the bound of the theorem.

So we restrict our attention henceforth to odd level vertices. For distinct odd i and j, level i vertices are not adjacent to level j vertices in D_V. Consequently, we could construct such a list for each odd level and concatenate the lists to achieve the desired list for all the odd

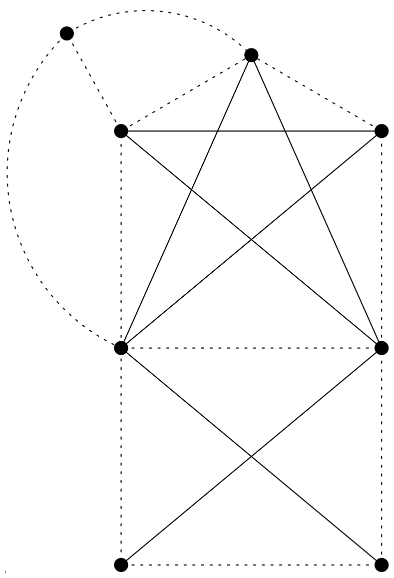

Figure 4: The edge dual (dotted lines) of Figure 2 and the corresponding vertex dual (solid and dotted lines).

vertices. In fact, we can restrict our attention even further to the level i vertices surrounded by a single level $i-1$ cycle in D_E, since vertices enclosed by two distinct level $i-1$ cycles in D_E cannot be adjacent and again we can simply concatenate their lists to obtain the desired list.

So consider the level i vertices within a cycle formed by level $i-1$ vertices in D_E. The subgraph of D_E induced by these vertices is an outerplanar graph $\overline{G} = (\overline{V}, \overline{E})$. The cliques that distinguish D_V from D_E may also include edges connecting some of these vertices in D_V, and these edges must be taken into account in constructing our lists.

Our plan is to construct a list for $\overline{V}$ in which each vertex v is adjacent in D_V to previously listed vertices from at most three cliques of D_V and to show that the previously listed vertices of two of these subsume the previously listed vertices of the third. The proof requires extensive analysis of the structure of faces in D_E. Each clique of D_V connects vertices of a face in D_E. A face of D_E that includes at least one vertex of $\overline{G}$ is either outside $\overline{G}$ or inside a face of $\overline{G}$. Henceforth, the terms outside and inside faces will denote these faces of D_E. Essentially, we handle the outside faces by choosing a good order in which to process faces of $\overline{G}$, and we handle the faces of D_E inside a face $\overline{F}$ of $\overline{G}$ by choosing a good order in which to list the vertices of $\overline{F}$.

First, we consider the outside faces; these include at least one vertex of $\overline{G}$ and lie outside

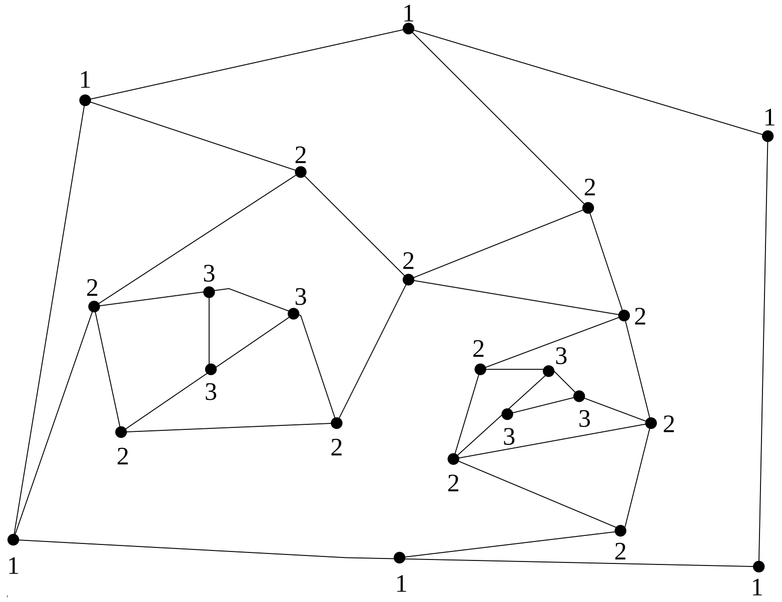

Figure 5: A level assignment for the vertices of the planar embedding of an edge dual.

$\overline{G}$. The complexity of the situation is illustrated in Figure 6. which shows a particular $\overline{G}$, together with the faces immediately outside it. A particular vertex v may be contained in more than one outside face. For example, a is contained in faces aCf, aCD, and $aDEb$. and g is contained in six outside faces in Figure 6. Each outside face includes one or more successive level $i-1$ vertices in the enclosing level $i-1$ cycle and one or more successive level i vertices of $\overline{G}$. Each such face is connected by a clique in D_V. In constructing our list for $\overline{V}$, the relevant clique edges are those connecting vertices of $\overline{V}$. Clique edges containing an endpoint at level $i-1$ do not affect adjacency within the list for $\overline{V}$ and can be ignored. Consequently, we can ignore any outside faces containing only one vertex of $\overline{V}$.

To deal with the outside faces of D_E, we construct a rooted ordered tree T representing the face connectivity of $\overline{G}$, as in the proof of Theorem 4. To construct the tree, we assume that $\overline{G}$ is connected. (If not, we pretend there are additional edges that connect it while preserving planarity.) Also, if there are any bridges, we consider them to be faces with two vertices and two (multi-)edges. Each vertex in T represents a face of $\overline{G}$, and the face represented by a vertex has at most two vertices in common with the face represented by any ancestor, any sibling, or the descendants of any sibling, The tree for $\overline{G}$ of Figure 6 is shown in Figure 7.

We use T to construct a list L of the vertices of $\overline{G}$. We do this recursively, starting with the root of T. For the face $\overline{F}$ corresponding to a vertex v of T, we list all as-yet-unlisted vertices of $\overline{F}$ in an order to be determined below, and then recurse on the children of v from left to

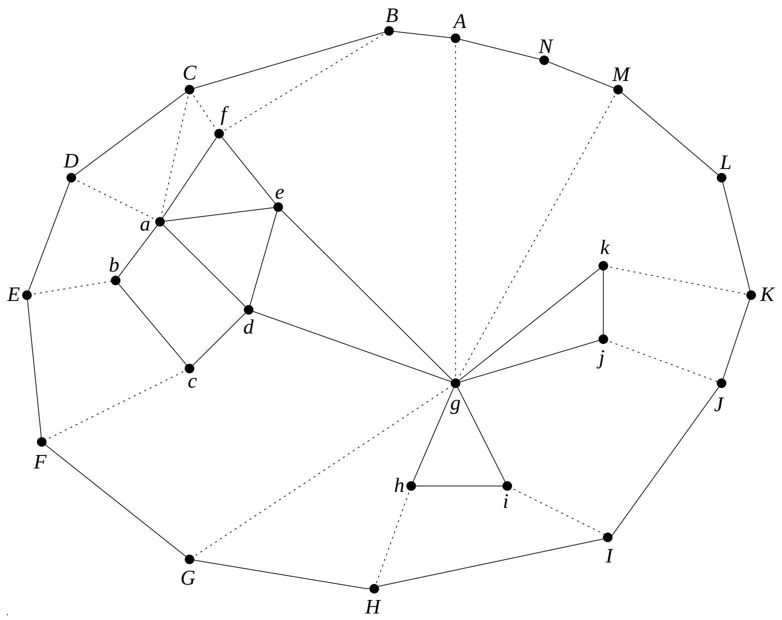

Figure 6: Faces outside a level i component $\overline{G}$. Edges between levels are represented as dotted lines.

right.

Consider the situation when a vertex v of $\overline{V}$ is listed. Since v is listed with the first face $\overline{F}$ of $\overline{G}$ containing it, level i vertices of at most two outside faces containing v have already been listed: one in the clockwise direction from v, and one in the counterclockwise direction from v.

The situation within $\overline{G}$ is more complicated. A face $\overline{F}$ of $\overline{G}$ corresponds to a level i cycle in D_E.

The vertices of $\overline{F}$ may belong to different inside faces (which we defined with respect to D_E), because of vertices and edges of higher level that are enclosed by $\overline{F}$ in D_E. Figure 8 illustrates a level i cycle and the faces within it in D_E. Since a single vertex of $\overline{G}$ can belong to multiple faces in D_E, we need to order our list of vertices in $\overline{G}$ based on the face connectivity of the inside faces, even though higher-level vertices are missing in $\overline{G}$.

In Figure 8, the face $(a, p, o, h, i, j, n, m, a)$ includes two paths h, i, j and m, a that lie on the level i cycle; in general, an inside face could have more than two paths that lie on the level i cycle and are separated by vertices not on the face. However, planarity prevents two inside faces from alternating paths; for example, in the level i cycle, there cannot be vertices belonging to face 1, then vertices belonging to face 2, then vertices belonging to face 1, and then vertices belonging to face 2.

Therefore, the intervals corresponding to inside faces are nested, and balanced parentheses

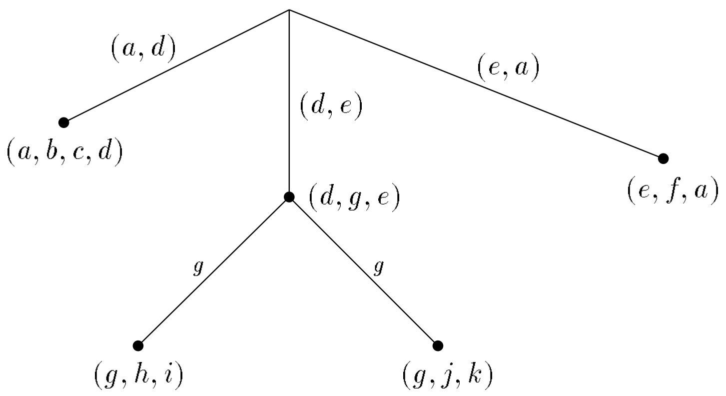

Figure 7: A tree T representing the face connectivity of the level i component $\overline{G}$ of Figure 6. Edges are labeled with the vertex or edge shared by the parent and child faces.

can be used to describe the nesting of intervals. For the example of Figure 8, if we begin and end at vertex a, we obtain $(a(ab(bc)(cdef)fg)(gh)(h)hij(jk)(kl)(lm)ma)$ for the level i cycle. A left parenthesis is used before the first vertex mentioned for each inside face in the counterclockwise traversal, and a right parenthesis is used after the last vertex of each face. Vertices are repeated in this list to represent inclusion in more than one inside face. This description is linear in the number of edges adjacent in D_E to vertices of $\overline{F}$.

We first list the vertices of the outermost level of parentheses and then recurse on the substrings within the next-outermost level of parentheses, while ignoring vertices already listed. Thus, for the above example, we list the vertices in the order $ahijmbfgcdekl$.

This method guarantees that of the vertices preceding a vertex v in L, those adjacent to v in D_V include only at most $c-1$ vertices belonging to a single inside face F_1 plus at most $2c-4$ vertices belonging to two outside faces F_2 and F_3. (Each outside face has at most $c-1$ vertices at level i.) Thus, we have a bound of $3c-5$ previously listed vertices of $\overline{F}$ adjacent to v. However, we will strengthen this bound to $2c-4$ as follows.

If the vertices of F_1 are subsumed by those of F_2 and F_3, the bound of $2c-4$ immediately follows. Otherwise, there is a vertex of F_1 not in F_2 or F_3, and by the parenthesization method and the contiguity of the vertices of F_2 and of F_3, for one of these outside faces, say F_3, the vertices of $F_3 - F_1$ are listed after v. Moreover, since v is listed with F_1 and not before, either F_1 is the first face listed, or another vertex of F_2 is also in F_1 and was listed, or F_2 contains only v from $\overline{F}$. Consequently, we obtain a bound of $2c-4$. $\square$

We observe that the schedule of Theorem 5 is always within 5 times the makespan of an

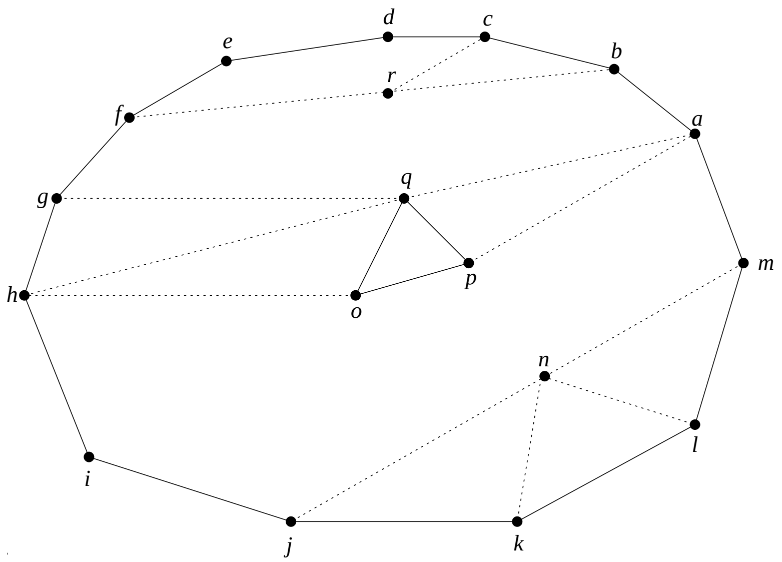

Figure 8: Face structure inside a level i face. Edges between levels are shown as dotted lines.

optimal schedule.

Also, we observe that the schedule of Theorem 5 can be computed in linear time using data structures as in [3] for planar embeddings. In particular, pointers are stored for each edge to identify the next edge clockwise and counterclockwise at each endpoint. With these data structures, the planar embedding of D_E can be constructed in linear time from the planar embedding of G and the levels, the decomposition into outerplanar graphs, and trees describing the face structure of the outerplanar graphs can be computed in time and space linear in the number of vertices as in [3]. Similarly, trees can be constructed in linear time to represent the parenthesized expressions for the faces inside each level i cycle. Recursing over these trees to construct the lists takes linear time. It is not necessary to explicitly construct D_V. Consequently, the entire computation of the list can be completed in time linear in the number of vertices, and as discussed earlier, GME constructs a schedule in time linear in the number of vertices as well.

4. Final Remarks

There are many interesting questions that remain open for MUTUAL EXCLUSION SCHEDULING. Further refinements of complexity would be desirable, especially for planar graphs. For example, is the optimization problem of Theorem 5 NP-complete? What is the complexity of BOUNDED INDEPENDENT SETS for planar graphs with m or t a fixed constant?

For both the two and three-dimensional domain decomposition problems, the worst-case bounds of our algorithms improve upon the worst-case bound of the standard coloring method. Since the standard coloring method does not necessarily perform at worst-case level, a superior algorithm would be to compute schedules using both methods and to take the better of the two schedules. It would be interesting to know if this approach offers a substantial improvement.

Finally, the generalization of MUTUAL EXCLUSION SCHEDULING to tasks of varying durations is of obvious interest; the complexity of number partitioning is added to the complexity of coloring in this more difficult problem. Bodlaender, Jansen, and Woeginger [7] have studied this generalization to the complementary problem of COMPATIBILITY SCHEDULING. They have worked out bounds on the performance of various approximation algorithms for graphs G having special structures.

References

[1] Noga Alon and Nabil Kahale. A spectral technique for coloring random 3-colorable graphs. In *Proc. 26th Annual ACM Symp. on the Theory of Computing*, pages 346–355, 1994.

[2] K. Appel and W. Haken. Every planar map is four colourable, part i: discharging. *Illinois J. Math.*, 21:429–490, 1977.

[3] Brenda S. Baker. Approximation algorithms for np-complete problems on planar graphs. *J. ACM*, 41(1):153–180, January 1994.

[4] B. Berger and J. Rompel. A better performance guarantee for approximate graph coloring. *Algorithmica*, 5:459–466, 1990.

[5] Petter Bjørstad, W. M. Coughran, Jr., and Eric Grosse. Parallel domain decomposition applied to coupled transport equations. In David E. Keys and Jinchao Xu, editors, *Domain Decomposition Methods in Scientific and Engineering Computing*, pages 369–380, 1995.

[6] H. L. Bodlaender and K. Jansen. Restrictions of graph partition problems, part i. Technical report ruu-cs-91-44, Department of Computer Science, Utrecht University, Utrecht, Netherlands, 1991.

[7] H. L. Bodlaender, K. Jansen, and G. J. Woeginger. Scheduling with incompatible jobs. *Discrete Applied Mathematics*, 55:219–232, 1994.

[8] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, New York, 1979.

[9] Frank Harary. *Graph Theory*. Addison-Wesley, Reading, Massachusetts, 1969.

[10] Z. Lonc. On the complexity of some chain and anti-chain partition problems. In *WG Conference*, pages 97–104, 1991.

[11] Carsten Lund and Mihalis Yannakakis. On the hardness of approximating minimization problems. In *Proceedings of the Twenty-Fifth Annual ACM Symposium on the Theory of Computing*, pages 286–293, 1993.

[12] D.J. Welsh and M.B. Powell. An upper bound for the chromatic number of a graph and its application to time-tabling problems. *Computer J.*, 10:85–86, 1967.