Scheduling with Conflicts

Sandy Irani* Vitus Leung†

Technical Report UCI-ICS-97-21

Department of Information and Computer Science

University of California, Irvine

Irvine, California 92697-3425

May 1997

Abstract

In this paper, we consider the scheduling of jobs that may be competing for mutually exclusive resources. We model the conflicts between jobs with a *conflict graph*, so that the set of all concurrently running jobs

*Supported in part by NSF Grants CCR-9309456 and GER-94-50142. Email: irani@ics.uci.edu .

†Supported in part by Caltrans Contract 65Q090. Email: vitus@ics.uci.edu .

must form an independent set in the graph. This model is natural and general enough to have applications in a variety of settings; however, we are motivated by the following two specific applications: traffic intersection control and session scheduling in high speed local area networks with spatial reuse. Our results focus on two special classes of graphs motivated by our applications: bipartite graphs and interval graphs. In all of the upper bounds, we devise algorithms which maintain a set of invariants which bound the accumulation of jobs on cliques (in the case of bipartite graphs, edges) in the graph. The lower bounds show that the invariants maintained by the algorithms are tight to within a constant factor. For the specific graph which arises in the traffic intersection control problem, we show a simple algorithm which achieves the optimal competitive ratio.

1 Introduction

In this paper, we consider scheduling jobs which are competing for limited resources. Jobs arrive in the system through time and require a certain set of resources to be completed. Any two jobs which require the same resource can not be executed simultaneously. We model the conflicts between jobs by a *conflict graph*, where each node in the graph represents a type of job. Jobs of the same type have the same requirements. If two types of jobs demand a common resource, there is an edge between those nodes in the graph. Thus, at all times, the set of jobs currently being executed must belong to nodes which form an independent set in the graph. Note that if there are two jobs of the same type in the system, one must wait until the other is completed.

We were motivated by the following two specific applications:

Traffic Intersection Control ([8, 11, 12, 15, 18, 19, 20, 26, 29, 31, 34, 35, 36, 38, 39]). Today's traffic intersection controllers are based on thirty year old signal phasing strategies. Signal phasings are optimized offline with historical data, downloaded into the controller and triggered by the presence of vehicles. Even the state of the art in adaptive traffic signal control only extend the optimization to a few seconds before every phase change. However, one expected consequence of an effective advanced traveler information system (ATIS) [1, 2, 3, 23] is the rerouting of congested traffic to streets and arterials that may either temporarily be under-utilized or which normally operate below capacity. Under such conditions, signal settings which have been determined based on recurrent traffic demand will not, in general, be "tuned" to accommodate the transient demand generated by the real-time driver information. As a result, system performance (as well as the effectiveness of the ATIS) is limited by the capacity of the signal system to adapt to transient traffic demand. Better strategies will be necessary for many of the proposed Intelligent Transportation Systems.

A traffic intersection is depicted in Figure 1. As all drivers know, the traffic on 1 is typically not allowed to proceed with the traffic on 2, 3, 4, 7, or 8. The complete conflict graph for the traffic intersection is also depicted in Figure 1. The intersection controller must schedule the vehicles through the intersection so as to avoid any conflicts. We consider a 'job' to be a platoon or closely spaced line of cars which must pass through the intersection.

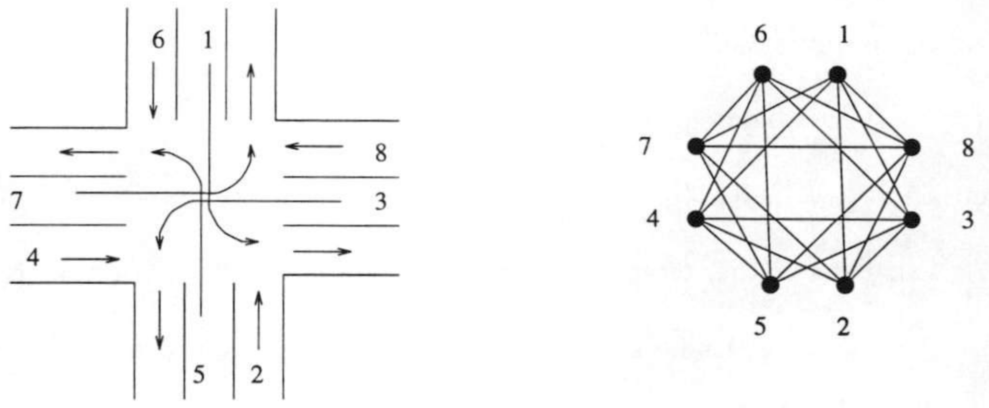

Figure 1: The graph depicting a traffic intersection.

Scheduling in high-speed local-area networks with spatial reuse ([14]). Local area networks with spatial reuse allow the concurrent access and transmission of user data with no intermediate buffering of packets. If some node s has to send data to some other node t, a 'session' is established between the s and t. A session typically lasts for much longer than its data transmission time and can be active only if it has exclusive use of all the links in its route from s to t. Therefore, sessions whose routes share at least one link are in conflict. Data transmissions among sessions must be scheduled so as to avoid these conflicts. We examine the problem of scheduling connections on a bus where there is exactly one possible route between any two pairs of points. Thus, if connections are defined by the two nodes which must be connected, it is determined whether a given pair of connections will conflict with each other.

In the first application and for small networks in the second application, it is reasonable to assume that each job requires roughly the same amount of time to execute. Thus, we adopt a discrete model of time and assume that each job requires one time unit to be completed once it is started. At the beginning of a time unit, jobs may arrive on any subset of the nodes in G; several jobs can arrive on the same node. The algorithm then chooses any independent set of nodes from which to schedule a job for each node. At the end of the time unit, the scheduled jobs are gone from the graph, and all other jobs remain on their respective nodes. Then at the beginning of the next time unit, another set of jobs may arrive.

There are two natural optimization problems that arise from this model. The first is to minimize the *total response time* of all jobs in the system. The second is to minimize the *maximum response time* of any job which enters the system. We focus on bounding the maximum response time of any job which is the maximum, over all jobs j, of $d_j - a_j$, where d_j is the time when j departs and a_j is the time when j arrives. In both applications we consider, it is important to guarantee the best turnaround time to any job entering the system. In the proofs, it will be convenient to refer to the *latency* of a job, which is the time that a job waits in the system before it is started. Since each job requires only one time unit to complete once it is started, the final latency of a job is one less than its response time.

1.1 Our Results

Typical scheduling algorithms are faced with the problem of making decisions about which jobs to perform at a given time without any knowledge of the demands that will be made on the system in the future. Such algorithms are said to be *online*. We compare the performance of the online algorithm to the performance of the optimal offline algorithm. An algorithm is said to be *c-competitive* if for any sequence of jobs, its cost on the sequence is at most c times the cost of the optimal offline algorithm on that sequence plus an additive term which is independent of the sequence. The *competitive ratio* of the algorithm is the infimum over all c such that the algorithm is c-competitive. In our results, we measure the performance of our algorithm by a generalized version of the competitive ratio in which we bound the cost of our algorithms by a (not necessarily linear) function of the adversary cost.

Minimizing the maximum response time for general conflict graphs is NP-hard, even when all jobs arrive in a single time unit: the problem is equivalent to graph coloring [22]. Even approximating the minimum maximum response time to within a fixed polynomial factor is NP-hard [27]. Our results focus on two special classes of graphs motivated by our applications: interval graphs and bipartite graphs. Such graphs can be optimally colored in polynomial time.

We argue in section 1.4 that the problem of scheduling with the traffic intersection conflict graph depicted in Figure 1 is equivalent to scheduling on a $K_{2,2}$. In section 3.1, we describe a simple algorithm and prove that it obtains a competitive ratio of 2 on a $K_{2,2}$. This result is then generalized in section 3.2 to arbitrary bipartite graphs. In section 3.3 we address scheduling where the conflict graph is an interval graph which models the problem of scheduling connections on a bus. An interval graph consists of a graph where each node can be identified with a closed interval on the real line. Two nodes are adjacent if and only if their corresponding intervals intersect. Consider a bus with processors $v_1, v_2, v_3, \ldots, v_n$ connected in order along the bus. Associate each processor with a point in the real line. The points are ordered from left to right as the processors are arranged on the bus. A connection between two nodes v_i and v_j for $i < j$ can be represented by an interval from the point just to the left of v_i to the point which is just to the right of v_j. Thus, two intervals intersect if and only if the two corresponding connections share a link.

Although the algorithms for bipartite and intervals graphs are quite different, the bounds they achieve are the same: we prove that for any sequence of jobs in which the maximum response time of a job in the optimal schedule is bounded by L, the algorithm can complete every job in time $O(n^3L^2)$, where n is the number of nodes in the conflict graph.

In all of the upper bounds, we devise algorithms which maintain a set of invariants which bound the accumulation of jobs on cliques (in the case of bipartite graphs, edges) in the graph. The lower bounds given in section 2 show that the invariants maintained by the algorithms are tight to within a constant factor. For a $K_{2,2}$, we describe an adversary which for any positive integer L and any algorithm, can devise a sequence which forces the algorithm to accumulate L jobs on an edge while the adversary has no jobs remaining in the graph. Furthermore, each job in the sequence can be completed by the optimal algorithm within L time units of its arrival. This bound gives a ratio of at least 2 for the competitiveness of any deterministic algorithm on a $K_{2,2}$ where the cost is the maximum response time of a job. Therefore, the simple algorithm described in section 3.1 achieves the optimal competitive ratio.

For bipartite and interval graphs, we describe an adversary which for any positive integer L and any algorithm, can devise a sequence which forces the algorithm to accumulate $\Omega(nL)$ jobs on a clique while the adversary has no jobs remaining in the graph. Again, each job in the sequence can be completed by the optimal algorithm within L time units of its arrival. These bounds give a lower bound of $\Omega(n)$ for the competitiveness of any deterministic algorithm on interval and bipartite graphs where the cost is the maximum reponse time of a job.

1.2 Previous Work

Over the past twenty-five years, most of the work done on scheduling with conflicts between certain pairs of jobs has been based on the well-known *Dining Philosophers* paradigm [4, 6, 13, 16, 17, 28, 32, 37] which is inherently a problem in decentralized control. More recently, Motwani, Phillips, and Torng [30] and Bar-Noy, Mayer, Schieber, and Sudan [7] considered problems with centralized control. In [30], each vertex in the conflict graph represents a single job and two vertices are adjacent when their corresponding jobs are in conflict. When jobs arrive at integral times and have unit execution time, Motwani *et al.* showed that the competitive ratio for the makespan is 2. When jobs arrive at arbitrary times and have arbitrary execution times, they gave an upper bound of 3 on the competitive ratio for the makespan using a model which allows for preemption. In [7], each vertex in the conflict graph represents a task that is to be scheduled as often as possible and again, two vertices are adjacent when their corresponding tasks cannot be scheduled concurrently. Bar-Noy *et al.* were interested in maximizing a measure of fairness among competing types. A number of authors have also considered a related static problem in multiprocessor scheduling [5, 9, 10, 21, 25].

1.3 Definitions

A node in a conflict graph G represents a class of jobs to be scheduled and two adjacent nodes in G represent two classes of jobs that cannot be scheduled together. Time is divided into discrete time units. At the beginning of a time unit, jobs may arrive on any subset of the nodes in G; several jobs can arrive on the same node. All jobs arrive with latency 0. The algorithm then chooses any independent set of nodes and schedules the oldest job waiting on each node in the independent set. At the end of the time unit, the scheduled jobs are gone from the graph and the latency of all the remaining jobs has increased by one. Then at the beginning of the next time unit, another set of jobs may arrive.

The *weight* of a node is defined to be the number of jobs waiting on the node, including the jobs that have just arrived in the system (i.e. the jobs with latency 0). A node is said to be *empty* if it has weight 0. The *weight* of an edge is the sum of the weights of the two endpoints.

1.4 The Traffic Intersection Graph

Notice that in the traffic intersection graph shown in Figure 1 since 1 and 2 are adjacent and have exactly the same neighborhood, they can be merged into one node. The same for 3 and 4, 5 and 6, 7 and 8. The resulting conflict graph is a $K_{2,2}$. Name the nodes of the $K_{2,2}$ r_1, r_2 on the right side and l_1, l_2 on the left side.

2 Lower Bounds

2.1 $K_{2,2}$

Lemma 1 *Let A be an arbitrary scheduling algorithm. If at the end of some time unit (say 0), the adversary has no jobs and A has i jobs on l_1 for some*

i less than L, then the adversary can force a configuration in which A has an edge of weight $i+1$ while the adversary has an empty graph, i.e. has no jobs left, and the adversary never has a job with latency more than $i+1$.

Proof. For up to $i+1$ consecutive time units, the adversary will have a job arrive on each of l_1 and r_1. For some $t \in \{1, 2, \ldots, i+1\}$, at the end of time unit t, the algorithm has $i+1$ jobs on l_1 or r_1. For any t in $\{1, 2, \ldots, i+1\}$, the adversary can schedule the jobs incurring a latency of at most $i+1$ so that after time t, l_1 or r_1 is empty.

The adversary strategy is the following. Keep offering jobs as described above until the algorithm has l_1 or r_1 with $i+1$ jobs after some time t. Assume without loss of generality that it is l_1. The adversary will serve the sequence up to that point so that l_1 is empty after time t.

This allows the adversary to adopt the following strategy to empty out its own graph while forcing the algorithm to keep $i+1$ jobs on edge (l_1, r_2). The adversary requests a job on r_2. The adversary schedules the oldest job on r_1 and the new job on r_2. After t time units, the adversary's graph is empty. Since a job was requested on r_2 in each time unit, the algorithm still has $i+1$ jobs on (l_1, r_2). ■

Theorem 2 *For $K_{2,2}$, no algorithm can have a competitive ratio for the maximum response time better than* 2.

Proof. There will be L stages. As j goes from 0 to $L-1$, we will start each stage with j jobs on l_1. We then invoke Lemma 1 to obtain an edge e with $j+1$ jobs. All we have to do is show how to establish the conditions of Lemma 1 for the next step. The adversary will do the following:

- For $2L$ consecutive time steps, have a job arrive on r_x, the right node incident to edge e. The adversary can schedule each job as it arrives. The algorithm must eventually have all $j+1$ jobs on r_x if it avoids having a job of latency $2L$.
- For $2L$ consecutive time steps have a job arrive on l_1. Eventually the algorithm must request all jobs on r_x and have $j+1$ jobs on l_1 if it avoids having a job of latency $2L$. The adversary schedules each job as it arrives.

At the end of the whole process, the algorithm has an edge (l_y, r_z) with L jobs while the adversary's graph is empty. For the next L time units, the adversary has a job arrive on r_z and l_y. By always scheduling the oldest job, the adversary never incurs a latency of more than L. The algorithm will have $2L$ jobs on edge (l_y, r_z) and must incur a latency of at least $2L$. ■

2.2 Bipartite and Interval Graphs

In order to prove the lower bound for bipartite and interval graphs, we prove the following lemma for a conflict graph consisting of a simple path on $4(k+1)$ nodes. Number every other edge from $-k$ to k, starting with the second edge. The rest of the edges are unnumbered. The nodes are numbered consecutively from $-2k-2$ to $2k+1$. The graph is pictured in Figure 2.

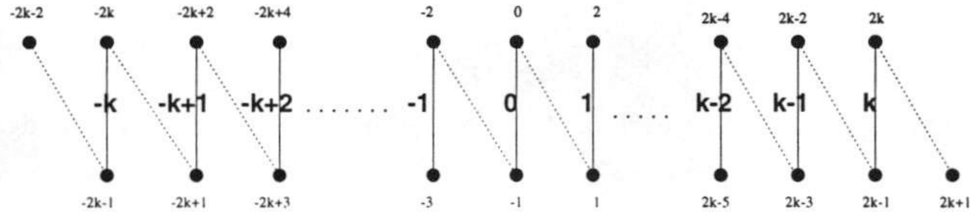

Figure 2: The conflict graph

Lemma 3 *Let A be an arbitrary scheduling algorithm. If at the end of some time unit (say 0), the adversary has no jobs and A has $Lk+i$ jobs on each odd node in the path for some i less than L, then the adversary can force a configuration in which A has an edge of weight $Lk+i+1$ while the adversary* *has an empty graph, and the adversary never has a job with latency more than L.*

Proof. For each $j = 0$ to k, the adversary will do the following for L consecutive time units:

- Have a job arrive on each of the odd numbered nodes incident to edges $-k, -k+1, \ldots, -(j+1)$.
- Have a job arrive on each of the two endpoints in edges $-j, -j+2, \ldots, j-2, j$.
- Have a job arrive on each of the even numbered nodes incident to edges $j+1, j+2, \ldots, k$.

Note that jobs are never placed on the first node $(-2k-2)$ or the last node $(2k+1)$. Figure 3 shows how the jobs arrive for $j = 2$ and $j = 3$ on a graph with $k = 5$.

Claim 4 *For some $t \in \{1, 2, \ldots, Lk + i + 1\}$, at the end of time unit t, the algorithm has $Lk + i + 1$ jobs on some unnumbered edge.*

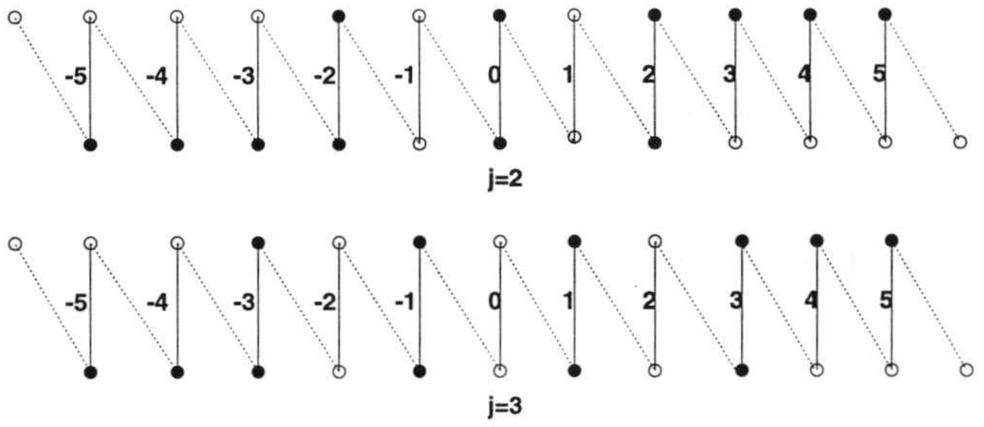

Figure 3: The job arrival pattern for a graph in which $k = 5$. Jobs arrive on the shaded nodes.

Proof of Claim 4. Let $t \in \{1, 2, \ldots, Lk + i + 1\}$. Let's assume that at the end of time t the algorithm has the following:

1. At least weight $Lk + i$ on each unnumbered edge.

2. Node $2k$ has weight at least t.

Note that the conditions are satisfied after time unit 0. After the jobs arrive at the beginning of time t, the weight of each unnumbered edge increases by 1. Furthermore a job always arrives on node $2k$.

If there is some unnumbered edge from which the algorithm does not schedule a job, then the weight of that edge remains $Lk + i + 1$ after the time unit, and we are done. Since node $-2k - 2$ (the first node in the path) never has any jobs on it, the only way for the algorithm to schedule a job from every unnumbered edge is to schedule a job from all the odd nodes. This would also maintain conditions 1 and 2. By the time $t = Lk + i + 1$, the second condition would imply that node $2k$ (and hence the last unnumbered edge) has weight at least $Lk + i + 1$. ∎

Claim 5 *Let e be any unnumbered edge. For any t in $\{1, 2, \ldots, Lk + i + 1\}$, the adversary can schedule the jobs incurring a latency of at most L so that after time t, e is empty.*

Proof of Claim 5. In the following scheme, the adversary can avoid having jobs on half the unnumbered edges. For each $0 \leq t \leq L(k+1)$, let

$$j = \left\lfloor \frac{t}{L} \right\rfloor .$$

During time t, the adversary will schedule the following.

1. The job that just arrived on each of the odd numbered nodes in edges $-k, -k+1, \ldots, -(j+1)$.

2. The oldest job on each of the even or odd numbered nodes in edges $-j, -j+1, \ldots, j$ depending on whether j is even or odd, respectively.

3. The job that just arrived on each of the even numbered nodes in edges $j+1, j+2, \ldots, k$.

If e is in the other half of the unnumbered edges then the adversary would use the same procedure except swap even and odd nodes in step 2. ■

The adversary strategy is the following. Keep offering jobs as described above until the algorithm has an unnumbered edge e with $Lk+i+1$ jobs at some time t. The adversary will serve the sequence up to that point so that its edge e is empty at time t. Notice also that the adversary's strategy in serving the requests has the property that at the end of a time unit, all even nodes are either empty or have the same set of jobs and all odd nodes are either empty or have the same set of jobs. Furthermore, if an odd node has x jobs, then all even nodes have at most $L-x$ jobs.

This allows the adversary to adopt the following strategy to empty out its own graph while forcing the algorithm to keep $Lk+i+1$ jobs on edge e. If the odd nodes in the adversary's graph have the oldest job, then the adversary requests a job on the odd node incident to e. The adversary schedules the oldest job on each odd node. Alternatively, if the even nodes in the adversary's graph have the oldest job, then the adversary requests a job on the even node incident to e. The adversary schedules the oldest job on each even node. After L time units, the adversary's graph is empty. Since a job was requested on e in each time unit, the algorithm still has $Lk + i + 1$ jobs on e. ∎

Theorem 6 *For bipartite graphs, no algorithm can have a competitive ratio for the maximum response time better than*

$$\frac{n}{4}.$$

Proof. Let the number of nodes n be $4(k+1)+1$ for some k. The graph consists of the path used in Lemma 3 with one additional node u adjacent to all the odd nodes in the path. There will be $k+1$ stages. As l goes from 0 to k, we will start each stage with Ll jobs on each of the odd nodes in the path from $-2l-2$ to $2l+1$. There are L steps within a stage. As j goes from 0 to $L-1$, we start each step with $Ll+j$ jobs on each of the odd nodes from $-2l-2$ to $2l+1$. We then invoke Lemma 3 to obtain an edge e with $Ll+j+1$ jobs. All we have to do is show how to establish the conditions of Lemma 3 for the next step. The adversary will do the following:

- For nL consecutive time steps, have a job arrive on v, the odd node incident to edge e. The adversary can schedule each job as it arrives.

The algorithm must eventually have all $Ll + j + 1$ jobs on v if it avoids having a job of latency nL.

- For nL consecutive time steps have a job arrive on u. Eventually the algorithm must request all jobs on v and have $Ll + j + 1$ jobs on u if it avoids having a job of latency nL. The adversary schedules each job as it arrives.

- If $j \leq L - 2$, let the set K be the nodes $-2l - 1, -2l + 1, \dots, 2l + 1$. Otherwise (we are setting up for a new stage), let K be the nodes $-2(l+1) - 1, -2(l+1) + 1, \dots, 2(l+1) + 1$. For the next nL time units, have a job arrive on each node in K. The adversary schedules all jobs as they arrive. The algorithm must eventually have $Ll + j + 1$ jobs on each node in K.

At the end of the whole process, the algorithm has an edge (v, w) with $L(k+1)$ jobs while the adversary's graph is empty. For the next L time units, the adversary has a job arrive on w and v. By always scheduling the oldest job, the adversary never incurs a latency of more than L. The algorithm will have $L(k+2)$ jobs on edge (v, w) and must incur a latency of at least $L(k+2) - 1$.

By plugging in

$$k = \frac{n-1}{4} - 1,$$

the algorithm's latency is at least

$$\frac{Ln}{4} - 1.$$

■

Theorem 7 *For interval graphs, no algorithm can have a competitive ratio for the maximum response time better than*

$$\frac{n}{4}.$$

Proof. The proof is exactly the same as the proof for bipartite graphs, except that the graph consists of a simple path on $4(k+1)$ nodes plus a node u which is adjacent to all nodes in the path. ■

3 Upper Bounds

3.1 $K_{2,2}$

The algorithm picks the side of the graph with the oldest job. If there is a tie, pick the side of the graph with the node of largest weight.

Lemma 8 *The algorithm will maintain the following invariants:*

1. *If the algorithm has an edge of weight w, then the adversary has at least $\max\{w - L, 0\}$ jobs on the edge.*

2. *If the algorithm has a node of weight w, then the adversary has at least $\max\{w - L, 0\}$ jobs on the node.*

Proof. Let us assume that the algorithm has maintained the invariants through time $t - 1$ and we will prove that the invariants are still maintained after time t. When new jobs arrive, clearly the invariants are still maintained. As long as the invariants are maintained, we will never have an edge of weight $2L + 2$ or more. Otherwise, the adversary would have an edge of weight $L + 2$ which would mean it eventually incurs a latency of at least $L + 1$ on some job. Thus, we know that it is impossible to have nodes of weight more than L on both sides of the graph. This means as long as the algorithm picks the side of the graph with the maximum weight node, there are no nodes of weight more than L on the other side of the graph and the invariants will be maintained after time t.

Now suppose that the algorithm does not pick the side of the graph with the maximum weight node. Assume without loss of generality that it is the left side and that l_1 is the node with the oldest job. We address two cases:

Case 1: l_1 has a job with latency at least L. The only problem happens if a node on the right side has at least $L+1$ jobs. Suppose r_1 is a node with at least $L+1$ jobs. (The same argument will hold for r_2 if it has at least $L+1$ jobs). Let's suppose the algorithm has x jobs on l_1 and y jobs on r_1. This means that the adversary has at least $x+y-L$ jobs on (l_1, r_1) at least $y-L$ of which must be on r_1. If the algorithm has any jobs on l_2, then a job will be taken from every edge, and invariant 1 will certainly be maintained. If there are no jobs on l_2, then the algorithm has y jobs on the edge (l_2, r_1) and we must be certain the adversary will continue to have at least $y-L$ jobs on the edge in order to maintain invariant 1. As long as we can verify that at least $y-L$ jobs remain on r_1, this condition as well as invariant 2 will be satisfied.

If the adversary still has the latency-at-least-L job on l_1, it must pick that one, which means that $y-L$ jobs still remain on r_1. Now suppose that the adversary does not still have the latency-at-least-L job on l_1. We first observe that no more than $x-1$ jobs have arrived on l_1 in the last $L-1$ time units (the algorithm has at most $x-1$ jobs that are more recent than the latency-at-least-L job and would not have picked a more recent job over the latency-at-least-L job). Thus, the adversary can have at most $x-1$ jobs on l_1, which means that it has at least $y-L+1$ jobs on r_1. Thus, at least $y-L$ remain after the time unit.

Case 2: l_1 does not have a job with latency at least L. Again, the only problem happens if a node on the right side has at least $L+1$ jobs. Suppose r_1 is a node with at least $L+1$ jobs. (The same argument will hold for r_2 if it has at least $L+1$ jobs). Let's suppose the algorithm has x jobs on l_1 and y jobs on r_1. This means that the adversary has at least $x+y-L$ jobs on (l_1, r_1) at least $y-L$ of which must be on r_1. If the algorithm has any jobs on l_2, then a job will be taken from every edge, and invariant 1 will certainly be maintained. If there are no jobs on l_2, then the algorithm has y jobs on the edge (l_2, r_1) and we must be certain the adversary will continue to have at least $y-L$ jobs on the edge in order to maintain invariant 1. As long as we can verify that at least $y-L$ jobs remain on r_1, this condition as well as invariant 2 will be satisfied.

We first observe that no more than $2L-1$ jobs can arrive on r_1 in $L-1$ consecutive time units. We next observe that no more than $L-1$ jobs on r_1 can be scheduled by the adversary during $L-1$ consecutive time units. Thus, at least $y-L$ remain after the time unit. ∎

Theorem 9 *The algorithm never has a job older than* $2L$*.*

Proof. Since the algorithm never has more than $2L+1$ jobs on an edge, when a job j arrives on a node (say l_1), there are never more than $2L$ other jobs on any edge incident to l_1. Let x be the number of jobs already on l_1 when the job j arrives. $0 \leq x \leq 2L$. There are at most $2L-x$ jobs on either node on the right side.

After the left side has been chosen x times, j will be the oldest job on the left side and will be scheduled the next time the left side is chosen. Thus, if we can prove that the right side is not chosen more than $2L-x$ times before j is scheduled, then we know that j will wait in the system at most $2L$ time units before it is scheduled. After the right side has been chosen $2L-x$ times, if j has not yet been chosen, then the left side has the oldest job in the system.

∎

Note that by using one greedy criteria, maximum latency, we are able to bound another greedy criteria, maximum weight. In general, this is not possible. On a path of length six, the algorithm is unbounded.

3.2 Bipartite Graphs

We will assume that we know L, the maximum latency incurred by the adversary. Let $X = (\frac{n}{2} + 4)L + 2$. Let H be the subgraph consisting of all edges whose weight is more than X and any node incident to such an edge. Suppose we have a simple path in the graph G and number the edges according to their sequence in the path, starting with 1. We say that the path is an *H-path* if all the odd numbered edges are in H. Call a node *immediately forbidden* if it is adjacent to a node of weight at least $X + 1$ and only has jobs with latency less than L. As soon as a node becomes immediately forbidden, it is *marked.* If there is a time unit in which there are no immediately forbidden nodes, then unmark all nodes. The set of *forbidden* nodes consists of all marked nodes which have at most one job with latency at least L.

The independent set is chosen in a series of three rounds:

Round 1: Pick all nodes that are adjacent only to empty nodes.

Round 2: Pick any node v such that there is an odd length H-path from a forbidden node to v.

Round 3: Pick the side of the graph with the oldest job and pick all nodes on that side of the graph which have not been already picked and are not adjacent to a node which has already been picked.

The algorithm will maintain the following invariants:

1. If the algorithm has weight $X + a$ on an edge, then the adversary has weight at least a on that edge.
2. If the algorithm has weight $X + a$ on a node, then the adversary has weight at least a on that node.
3. There is no odd length H-path between two forbidden nodes.

Lemma 10 *The algorithm picks an independent set.*

Proof. The only way for two adjacent nodes, w_1 and w_2, to be picked is if there are two forbidden nodes, v_1 and v_2, such that there is an odd length H-path from v_1 to w_1 and from v_2 to w_2. This means that there is an odd length H-path from v_1 to v_2 which violates the last invariant. ■

Lemma 11 *The algorithm maintains the invariants.*

Proof. We first prove that invariant 3 holds as a result of invariant 1. Consider a forbidden node v; v has at most one job older than L. Any node can have at most $2L$ jobs with latency less than L because if more than $2L$ jobs arrive at the same node during L consecutive time units, then the adversary latency is more than L. Thus, v has weight at most $2L + 1$.

Now consider a simple path with an odd number of edges and an even number of nodes, $p_0, p_1, \ldots, p_{2k+1}$. Suppose that p_0 and p_{2k+1} are both forbidden and that for each $0 \leq i \leq k$, the edge (p_{2i}, p_{2i+1}) is in H (i.e. invariant 3 is false). Consider three consecutive nodes in the path $p_{2i-2}, p_{2i-1}, p_{2i}$. If the weight of p_{2i-2} is x, then the weight of p_{2i-1} is at least $X + 1 - x$, since the edge (p_{2i-2}, p_{2i-1}) is in H. But then the weight of node p_{2i} is at most $x + L$ since the edge (p_{2i-1}, p_{2i}) has weight at most $X + L + 1$. Thus, the weight of the even nodes in the path increases by at most L along the path. Since p_{2k+1} has weight at most $2L+1$, p_{2k} has weight at least $X - 2L$. Furthermore, since p_0 has weight at most $2L + 1$, there must be at least

$$\left\lceil \frac{(X - 2L) - (2L + 1)}{L} \right\rceil > \frac{n}{2}$$

even nodes in the path. Since there are the same number of even nodes and odd nodes in the path, there are not enough nodes in the graph.

Next we argue that if invariant 1 holds, then invariant 2 must hold as well. Suppose there is a node v for which invariant 2 does not hold. If there were a node w adjacent only to v which never received a job, the behavior of the algorithm (and adversary) would be identical. So if invariant 2 does not hold for v, then invariant 1 would not hold for the edge (v, w).

We now prove that invariant 1 holds for each edge (v, w) in H. We address two cases:

Case 1: There are jobs waiting on both w and v. Let u be a neighbor of v. If u is picked in round 2, then w is also picked since that path u, v, w would be a continuation of an odd length H-path to u. The same holds for a neighbor of w. Thus, if neither v or w is picked in round 2, neither is adjacent to a node which has been picked and one of them will be picked in round 3.

Case 2: There are no jobs on w. (The same argument holds if there are no jobs on v). Thus, v must have weight at least $X + 1$. If v has a neighbor all of whose jobs have latency less than L, then v is adjacent to a forbidden node and would have been picked in round 2. Furthermore, if it is adjacent to only empty nodes, it will be chosen in round 1.

It remains to address the case where v has weight at least $X+1$ and is adjacent to some node u with a job of latency at least L. Let's say that v has weight $X+a$ and u has weight b. This means that the adversary has at least $a+b$ jobs on (v,u), at least a of which must be on v.

Let b' be the number of jobs that have arrived on node u in the last $L-1$ time units. We know that the algorithm has all b' of these jobs still on u since the older latency-at-least-L job is still remaining. Thus, the b jobs which the algorithm has on u include the latency-at-least-L job and the b' jobs which arrived since, and we can conclude that $b' \leq b-1$.

If the adversary has the latency-at-least-L job on u, it must pick it in the next time step in which case a of its jobs will remain on v. If the adversary does not have the latency-at-least-L job, then it has at most $b' \leq b-1$ jobs on u. In this case, the adversary has at least $a+1$ jobs on v and at least a will remain after the current step. ■

Theorem 12 *The maximum response time of a job is $O(n^3L^2)$.*

Proof. When a job j arrives, there are at most $n(X+L+1)$ other jobs in the system. We will argue that every $Ln+1$ time units, there is a time when there are no forbidden nodes. This implies that every $Ln+1$ time units, at least one job that is older than j gets eliminated. Thus after $n(X+L+1)(Ln+1) = O(n^3L^2)$ time units, j will be scheduled.

A node can be immediately forbidden for at most L consecutive time units after which it will remain marked and can not be immediately forbidden again until it is unmarked. Nodes only become unmarked during a time unit in which there are no immediately forbidden nodes. After Ln consecutive time units in which there is an immediately forbidden node, all nodes will be marked and there will be no immediately forbidden nodes. ■

We will use a doubling trick to remove the assumption that we know the maximum optimal latency in advance. We 'guess' L' and start with a guess of $L' = 1$. Notice that as long as the invariants are maintained, the maximum latency of a job will be $O(L'^2n^3)$. If ever the invariant is violated for L', we double our guess for L' and continue. Note that when L' doubles, the invariants are still guaranteed to be maintained. Furthermore, the optimal schedule has a job with latency which is at least half of the algorithm's current guess.

3.3 Interval Graphs

We will call a node in the graph and the closed interval on the real line which defines the node by the same name. For an interval v, let $l(v)$ be its left endpoint and $r(v)$ be its right endpoint. Without loss of generality, we can assume that for any node, the left endpoint and the right endpoint are distinct. Consider a point p on the real line. We call the *weight* of point p to be the sum of the weights of the nodes whose intervals contain p. We will initially assume that the algorithm knows that the maximum adversary latency is at most L. Let $X = (2L + 1)n$. The algorithm will maintain the following invariant:

- If the algorithm has weight $X + a$ on p, then the adversary has weight at least a on p.

Note that since the adversary never has a job with latency more than L, it never has a point with weight more than $L + 1$. By the invariant, this means that the algorithm never has a point with weight more than $X + L + 1$.

The set of all points of weight $X + j$ forms a finite set of disjoint intervals (some closed, some open, some half-open). We will call this set of intervals $\mathcal{I}_j$. The set of all points of weight at least $X + 1$ form a finite set of disjoint closed intervals which we will call $\mathcal{I}$. Consider the set of points which are an endpoint of an interval in $\mathcal{I}_j$ for some $1 < j \leq L + 1$ but not an endpoint of an interval in $\mathcal{I}$. Such a point must be the endpoint of a node of weight at most L because the point has weight at most $X + L + 1$ and the point just to its left or right has weight at least $X + 1$. Let S be the subset of these points which are an endpoint of a node with a job of latency at least L. Let $\mathcal{I}'$ be the set of all intervals which are in $\mathcal{I}_j$ for some $j \geq 1$ and are contiguous with a point in S. Finally, consider the set of all points which are in some interval in $\mathcal{I}$ but not some interval in $\mathcal{I}'$. When these points are grouped in maximal contiguous intervals, they form a finite set of disjoint intervals. We will call this set of intervals $\mathcal{J}$ and will name them from left to right: $J_1, J_2, \ldots, J_m$. Note that some of the intervals may be open or half-open. Since it is more convenient to talk about closed intervals, we will add endpoints if necessary to intervals in $\mathcal{J}$ to assure that they are all closed. The algorithm will pick an independent set which covers $\mathcal{J}$. We will then prove that this is sufficient to maintain the invariant.

Before the end of each time step, we must pick an independent set of nodes. To do this, we will define a set of *forbidden* nodes. Once this set has been determined, we can greedily pick an independent set from the set of non-forbidden nodes. In order to determine the set of forbidden nodes, we require some definitions. A node is *immediately forbidden* if it only has jobs of latency less than L. We *mark* every node when it becomes immediately forbidden. A node is said to be *eligible* if it has jobs and has a non-empty intersection with an interval in $\mathcal{J}$.

At the beginning of a given time unit, if there are no immediately forbidden nodes, then we unmark all nodes. The set of forbidden nodes in this time unit is then empty. Otherwise, we initialize the set of forbidden nodes to be all the marked nodes which have at most one job older than L. We then make a pass through the graph from right to left considering each interval J_i in reverse order (as i goes from m to 1). Let J be the current interval under consideration, and let w be the node with the right-most left endpoint whose interval includes $l(J)$ and is not forbidden. Add to the forbidden set all eligible nodes whose right endpoint is in the interval $[l(w), l(J)]$. (Refer to Figure 4).

Lemma 13 *For any point p on the real line, the total weight of forbidden nodes which intersect p is at most $X = (2L+1)n$.*

Proof. We will scan the real line from right to left keeping track of the maximum weight of forbidden nodes seen so far on any point. Every time the

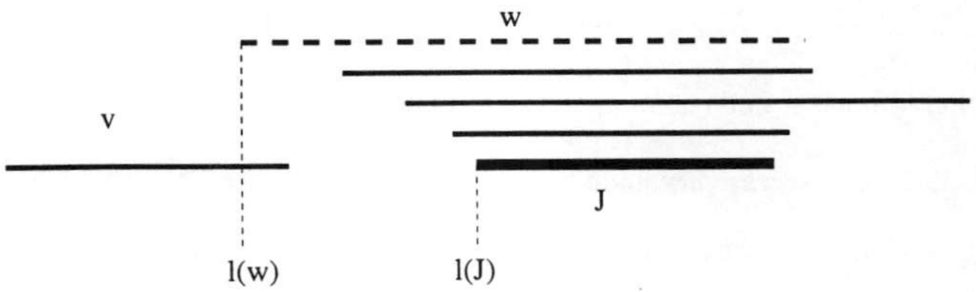

Figure 4: The forbidden nodes are shown in solid. The unforbidden node is shown in a dashed line. The interval J must be covered by some node; v becomes forbidden because its right endpoint lies in $[l(w), l(J)]$

maximum weight of forbidden nodes increases, we will attribute the increase to the right endpoint of some node. Furthermore, we will show that the maximum weight of forbidden nodes increases at each point by at most $2L+1$.

Now suppose that the maximum weight of forbidden nodes we have seen so far increases when the right endpoint of a forbidden node v has been reached. Let the previous maximum be x. If v was added to the forbidden set when it was initialized, then v has at most one job older than L. Any node can have at most $2L$ jobs with latency less than L because if more than $2L$ jobs arrive during L consecutive time units, then the adversary latency is more than L. Thus, v has weight at most $2L+1$ and we can attribute the increase caused by v to the right endpoint of v.

Now suppose that v was added because there are no unforbidden nodes whose left endpoint falls in the interval $(r(v), l(J)]$ for some interval $J \in \mathcal{J}$. Now consider the point just to the right of $l(J)$. We know that at this point, the weight is at least $X + 1$ and the weight of the forbidden nodes is at most x. Thus, the weight of the non-forbidden nodes at $l(J)$ is at least $X + 1 - x$. Since none of these unforbidden nodes have left endpoints in $(r(v), l(J)]$, the weight of the unforbidden nodes at $r(v)$ must be at least $X + 1 - x$. Since the weight of $r(v)$ is at most $X + L + 1$, the weight of forbidden nodes at $r(v)$ is at most $x + L$. We can account for the increase from x to $x + L$ by the right endpoint of v. ■

Once the set of forbidden nodes has been determined, we can pick the independent set as follows. If there are no forbidden nodes in the graph, start with the node v with the oldest job. Scan right from v and then left from v, greedily picking the independent set (i.e., whenever the end of the current node is reached, pick the next node reached). If there are forbidden nodes then scan from left to right. Pick the first eligible unforbidden node that is reached. When the right endpoint of that node is reached, pick the eligible unforbidden node with the next left endpoint. Continue until the end of the intervals have been reached. The fact that the invariant is maintained is implied by the following two lemmas.

Lemma 14 *Let p be any point contained in an interval in $\mathcal{I}'$. If the algorithm has weight at least $X + a$ on p at the beginning of the time unit, then the adversary will have weight a on p at the end of the time unit.*

Lemma 15 *The independent set chosen by the algorithm covers $\mathcal{J}$.*

Proof of Lemma 14. Consider some interval $I \in \mathcal{I}'$. I is an interval in $\mathcal{I}_j$ for some $j \geq 1$ which was placed in $\mathcal{I}'$ because I is contiguous with a point p in S. Let Q be the set of all nodes which cover p and do not cover all of I. Since the interval I has the same weight on every point, the algorithm does not have any jobs on any node with an endpoint in I. Suppose that the sum of the weights of all nodes in Q is x and the weight of interval I is y. The algorithm has weight $x + y$ on p. The adversary has at least weight $x + y - X$ on p and at least weight $y - X$ on nodes which cover all of I.

The algorithm has x jobs on nodes in Q, at least one of which has latency at least L. If the adversary has any latency-L jobs on any node in Q, it must schedule that job and will continue to have weight $y - X$ on all of I. If the adversary does not have any latency-L jobs on Q, then it must have weight at most $x - 1$ on Q. This is due to the fact that since the algorithm does not schedule any jobs of latency less than L, if the adversary has any such jobs, then the algorithm also has them. Thus, in this case, the adversary has weight at most $x - 1$ on Q which means it must have weight at least $y - X + 1$ on all of I. At least $y - X$ of these jobs will remain after the time unit. ■

Proof of Lemma 15. Consider any node v with jobs such that the point just to the right of $r(v)$ or just to the left of $l(v)$ is in J_i for some $J_i \in \mathcal{J}$. Let's say it is the point just to the right of $r(v)$. If v has any jobs, then the weight at $r(v)$ must be greater than the weight just to the right of $r(v)$. In this case, $r(v)$ is the endpoint of some interval in $\mathcal{I}_j$ for some $j > 1$ but is not an endpoint of an interval in $\mathcal{I}$. (Since $\mathcal{I}$ contains all of $\mathcal{J}$, any point in $\mathcal{J}$ is also in $\mathcal{I}$). If $r(v) \in S$, then the point just to the right of $r(v)$ is in $\mathcal{I}'$ and is excluded from $\mathcal{J}$ (a contradiction). If $r(v)$ is not in S, then all its jobs must have latency less than L. This means that v is immediately forbidden.

We start with the case where there are no forbidden nodes. When we start with the node with the oldest job and scan the graph left and right from there, greedily picking the independent set, we know that there will always be some node to pick before we hit an interval in $\mathcal{J}$ (otherwise there would be a node

v such that v has jobs and the point just to the right of $r(v)$ or just to the left of $l(v)$ is in J_i for some $J_i \in \mathcal{J}$). Furthermore, we are guaranteed that if a node intersects some interval in $\mathcal{J}$, it will cover the interval (otherwise, again, by the above argument it would have been immediately forbidden).

If there are forbidden nodes, then by Lemma 13, the left endpoint of the first unforbidden node is reached before any point of weight $X + 1$ is reached (since the maximum weight of forbidden nodes is at most X). Therefore, there are no points in $\mathcal{J}$ to the left of the independent set. Now suppose that there is a point in $\mathcal{J}$ which is not covered by the independent set. Let p be the left-most such point. Let v be the node in the independent set closest and to the left of p. If p is not the left endpoint of an interval in $\mathcal{J}$, then by the argument at the beginning of this proof, v would have been immediately forbidden. Thus, we can assume that $p = l(J)$ for some $J \in \mathcal{J}$. There must be no eligible unforbidden nodes whose left endpoint falls in the range $(r(v), l(J)]$, otherwise they would have been picked in the independent set and $l(J)$ would have been covered. But in this case, v would have been forbidden. ■

Theorem 16 *The maximum response time of a job is $O(n^3L^2)$.*

Proof. The proof is identical to the proof of Theorem 12. ■

We will use the same doubling trick used with interval graphs (and described at the end of the previous section) to remove the assumption that we know the maximum optimal latency in advance. The trick works for the same reasons it worked for interval graphs.

During each time step, the complete algorithm is as follows:

- Update our guess for L.
- Determine the sets of intervals $\mathcal{I}_j$ for $1 \leq j \leq A+1$.
- Determine the set of points S.
- Determine the set of intervals $\mathcal{I}'$.
- Determine the set of intervals $\mathcal{J}$.
- Determine the set of forbidden nodes.
- Pick an independent set of nodes and execute a job from each of those nodes.

4 Open Problems

Even though the lower bounds for bipartite and interval graphs show that the invariants maintained by the algorithms, i.e. the number of jobs on a node or clique in excess of the adversary, are tight to within a constant factor, there is still quite a gap in the bounds for maximum response time. In particular, is it possible to obtain a competitive ratio for either bipartite or interval graphs which is strictly a function of n?

Modeling the problem of scheduling connections on other local area network architectures would require the use of more general classes of intersection graphs (e.g. tree graphs, the intersection graphs of paths in a tree [33]). Is it possible to obtain bounds on more general classes of intersection graphs? Alternatively, is there a graph for which there is no competitive algorithm?

Finally, the assumption that the input sequence of job arrivals can be arbitrarily bad may be somewhat pessimistic. An interesting direction for future research would be to restrict the power of the adversary by allowing only certain input distributions as proposed by Koutsoupias and Papadimitriou [24].

References

[1] J.L. Adler, W.W. Recker and M.G. McNally, Using interactive simulation to model driver behavior under ATIS. in *Proceedings for the 4th International Conference on Microcomputers in Transportation* (J. Chow, ed.). ASCE. New York, 1993, pp. 344–355.

[2] H. Al-Deek, S. Ishak and A.E. Radwan, The potential impact of advanced traveler information systems (ATIS) on accident rates in an urban transportation network. in *Proceedings for the 4th Vehicle Navigation and Information Systems Conference.* IEEE. Piscataway, NJ, 1993, pp. 633–636.

[3] H. Al-Deek and A. Kanafani, Modeling the benefits of advanced traveler information systems in corridors with incidents. *Transportation Res. Part C*, **1** (1993), 303–324.

[4] B. Awerbuch and M. Saks, A dining philosophers algorithm with polynomial response time, in *Proceedings for the 31st Symposium on the Foundations of Computer Science.* IEEE Computer Society. Washington, DC, 1990, pp. 65–74.

[5] B.S. Baker and E.G. Coffman, Jr., Mutual exclusion scheduling, *Theoret. Comput. Sci.*, **162** (1996), 225–243.

[6] J. Bar-Ilan and D. Peleg, Distributed resource allocation algorithms, in *Proceedings for the 6th International Workshop on Distributed Algorithms* (A. Segall and S. Zaks, eds.). Springer-Verlag. New York, 1992, pp. 277–291.

[7] A. Bar-Noy, A. Mayer, B. Schieber and M. Sudan, Guaranteeing fair service to persistent dependent tasks, in *Proceedings for the 6th Symposium on Discrete Algorithms.* SIAM. Philadelphia, 1995, pp. 243–252.

[8] M. Bell, Future directions in traffic signal control, *Transportation Res. Part A*, **26** (1992), 303–313.

[9] H.L. Bodlaender and K. Jansen, On the complexity of scheduling incompatible jobs with unit-times, in *Mathematical Foundations of Computer Science* (A.M. Borzyszkowski and S. Sokolowski, eds.). Springer-Verlag. Berlin, 1993, pp. 291–300.

[10] H.L. Bodlaender, K. Jansen and G.J. Woeginger, Scheduling with incompatible jobs, *Discrete Appl. Math.*, **55** (1994), 219–232.

[11] R.D. Bretherton and G.T. Bowen, Recent enhancements to SCOOT–SCOOT version 2.4, in *Proceedings for the 3rd International Conference on Road Traffic Control.* IEE, London, 1990, pp. 95–98.

[12] D. Bullock and C. Hendrickson, Roadway traffic control software. *IEEE Trans. Control Systems Technology,* **2** (1994), 255–264.

[13] K. Chandy and J. Misra, The drinking philosophers problem, *ACM Trans. Programming Lang. and Systems,* **6** (1984), 632–646.

[14] J. Chen, I. Cidon and Y. Ofek, A local fairness algorithm for gigabit LANs/MANs with spatial reuse, *IEEE J. Selected Areas in Comm.,* **11** (1993), 1183–1192.

[15] S. Chiu and S. Chand, Adaptive traffic signal control using fuzzy logic, in *Proceedings for the 2nd IEEE International Conference on Fuzzy Systems.* IEEE, New York, 1993, pp. 1371–1376.

[16] M. Choy and A.K. Singh, Efficient fault tolerant algorithms for resource allocation in distributed systems, in *Proceedings for the 24th Symposium on the Theory of Computing.* ACM. New York, 1992, pp. 593–602.

[17] E.W. Dijkstra, Hierarchical ordering of sequential processes, *Acta Inform.*, **1** (1971), 115–138.

[18] R.E. Fenton, IVHS/AHS: driving into the future, *IEEE Control Systems Mag.*, **14**:6 (December 1994), 13–20.

[19] N.H. Gartner, OPAC: strategy for demand-responsive decentralized traffic signal control, in *Control, Computers, Communications in Transportation* (J.P. Perrin, ed.). Pergamon. Oxford, UK, 1990, pp. 241–244.

[20] J.J. Henry and J.L. Farges, PRODYN in *Control, Computers, Communications in Transportation* (J.P. Perrin, ed.). Pergamon. Oxford, UK, 1990, pp. 253–255.

[21] K. Jansen, Scheduling of incompatible jobs on unrelated machines, *Internat. J. of Found. of Comput. Sci.*, **4** (1993), 275–291.

[22] R.M. Karp, Reducibility among combinatorial problems, in *Complexity of Computer Computations* (R.E. Miller and J.W. Thatcher, eds.). Plenum. New York, 1972, pp. 85–103.

[23] A.J. Khattak, J.L. Schofer and F.S. Koppelman, Commuters' enroute diversion and return decisions: analysis and implications for advanced

traveler information systems. *Transportation Res. Part A*, **27** (1993), 101–111.

[24] E. Koutsoupias and C.H. Papadimitriou, Beyond competitive analysis, in *Proceedings for the 35th Symposium on the Foundations of Computer Science.* IEEE Computer Society. Washington, DC, 1994, pp. 394–400.

[25] J.K. Lenstra, D.B. Shmoys and E. Tardos, Approximation algorithms for scheduling unrelated machines, *Math. Programming*, **46** (1990), 259–271.

[26] P.R. Lowrie, *SCATS, Sydney Co-Ordinated Adaptive Traffic System: a Traffic Responsive Method of Controlling Urban Traffic*, Roads and Traffic Authority, Darlinghurst, NSW, Australia, 1990.

[27] C. Lund, M. Yannakakis, On the hardness of approximating minimization problem, in *Proceedings for the 25th Symposium on the Theory of Computing.* ACM. New York, 1993, pp. 286–293.

[28] N. Lynch, Upper bounds for static resource allocation in a distributed system, *J. Comput. System Sci.*, **23** (1981), 254–278.

[29] V. Mauro and C. Di Taranto, UTOPIA in *Control, Computers, Communications in Transportation* (J.P. Perrin, ed.). Pergamon. Oxford, UK, 1990, pp. 245–252.

[30] R. Motwani, S. Phillips and E. Torng, Non-clairvoyant scheduling, in *Proceedings for the 4th Symposium on Discrete Algorithm.* SIAM. Philadelphia, 1993, pp. 422–431.

[31] C. Pappis and E. Mamdani, A fuzzy logic controller for a traffic junction, *IEEE Trans. Systems Man Cybernet.*, **7** (1977), 707–717.

[32] M. Rabin and D. Lehmann, On the advantages of free choice: a symmetric and fully distributed solution to the dining philosophers problem, in *Proceedings for the 15th Symposium on Principles of Programming Languages.* ACM New York, 1988, pp. 133–138.

[33] P.L. Renz, Intersection representations of graphs by arcs, *Pacific J. Math.*, **34** (1970), 501–510.

[34] F.S. Roberts, *Discrete Mathematical Models with Applications to Social Biological, and Environmental Problems*, Prentice Hall, Englewood Cliffs, NJ, 1976.

[35] D.I. Robertson and R.D. Bretherton, Optimizing networks of traffic signals in real time–the SCOOT method, *IEEE Trans. on Vehicular Tech.*, **40** (1991), 11–15.

[36] K.E. Stoffers, Scheduling of traffic lights - a new approach, *Transportation Res.*, **2** (1968), 199–234.

[37] E. Styer and G. Peterson, Improved algorithms for distributed resource allocation, in *Proceedings for the 7th Symposium on Principles of Distributed Computing*. ACM. New York, 1988, pp. 105–116.

[38] F.V. Webster, *Traffic Signal Settings*, Her Majesty's Stationery Office, London, 1958.

[39] R.L. Wilshire, Traffic signals, in *Traffic Engineering Handbook* (J.L. Pline, ed.). Prentice Hall. Englewood Cliffs, NJ, 1992, pp. 278–309.