SCHEDULING OF VIRTUAL CONNECTIONS IN FAST NETWORKS

THOMAS ERLEBACH
Institut für Informatik, Technische Universität München,
D–80290 München, Germany
ERLEBACH@INFORMATIK.TU-MUENCHEN.DE

KLAUS JANSEN
FB IV – Mathematik, Universität Trier, Postfach 3825,
D–54286 Trier, Germany
JANSEN@DM3.UNI-TRIER.DE

In communication networks based on ATM it is necessary to allocate sufficient resources whenever a new connection is established. In particular, a certain amount of bandwidth must be reserved along a path between sender and receiver. Several *virtual* connections can share a physical link if the sum of their bandwidths does not exceed the link capacity. Given a network and a set of connection requests, one is interested to find a schedule for these calls that minimizes the completion time. For the restricted setting with unit call durations and unit bandwidth requirements, it is shown that the call scheduling problem is $\mathcal{NP}$–complete in tree networks of arbitrary degree and in ring networks. A polynomial-time optimal algorithm for bidirectional calls in binary trees and a 1.1–approximation algorithm for bidirectional calls in arbitrary trees are given. In addition, $\mathcal{NP}$–completeness is shown for calls with arbitrary duration or arbitrary bandwidth requirements in a wide class of networks.

1 Introduction

In recent years new network technologies and communication protocols have emerged. ATM[2] (*Asynchronous Transfer Mode*) is expected to replace Ethernet and FDDI technology in local area networks in the near future. This will improve the communication performance in workstation clusters and increase their usefulness for parallel computation. ATM will also be used in wide area networks, thereby solving many problems that are currently related to the use of different technology and protocols in local and long-distance communication. Furthermore, all-optical networks employing wavelength-division multiplexing (as opposed to time-division multiplexing used in ATM networks) have lately received increased attention.

One of the characteristics of these new networks is the notion of guaranteed quality of service. When a connection between sender and receiver is established, the network guarantees that the specified bandwidth will be available to the connection as long as it remains active. In order to make such performance guarantees possible, the network has to allocate sufficient resources to each established connection. More precisely, in an ATM network sufficient buffers and bandwidth must be reserved in all ATM switches along a physical path between sender and receiver. In all-optical networks, each connection must be assigned a physical path in the network and a wavelength such that no two connections using the same link are assigned the same wavelength. If it is not possible to establish all requested connections at the same time because of resource limitations, some connections can be deferred and established later when other connections have already terminated.

Given a set of connection requests, one is interested to determine the minimum time necessary to complete all connections. We refer to this problem as *call scheduling* and we let the word *call* stand for both "connection" and "connection request." If the requests arrive on-line and the scheduling algorithm does not have any knowledge about future requests or the durations of calls, the quality of an algorithm can be determined by *competitive analysis*. In the off-line scenario, where the algorithm has complete knowledge about future requests and call durations, the exact optimization problem is $\mathcal{NP}$–hard in most instances. Therefore, one is interested in good approximation algorithms.

1.1 Model

The communication network is represented by a directed graph $G = (V, E)$ with $(e = (u, v) \in E) \Rightarrow (\hat{e} := (v, u) \in E)$. Each edge e has an associated capacity $c(e)$, satisfying $c(e) = c(\hat{e})$. Intuitively, a bidirectional link of the physical network is represented by two directed edges with the same capacity.

The set of call requests is denoted by R, and each request $r \in R$ has the form $r = (u, v, b, d)$. Here, u and v are the source and target node of the call, respectively, b is the requested bandwidth, and d is the duration.

In this paper, we assume that all call requests are available at time 0 and do not have to be completed by a certain deadline. There is no precedence relation among the calls. We consider the off-line version of the problem, where call durations are known in advance. In section 2 and 3 we restrict our attention to the case that all edges have unit capacity, all calls require unit bandwidth, and all call durations are the same (unit duration). In section 4 we present some results for the more general setting of arbitrary call durations and arbitrary bandwidth requirements.

A feasible schedule assigns to each call request $r = (u, v, b, d)$ a directed path from u to v in G and a starting time t_r such that the sum of bandwidths of simultaneously active calls using an edge e does not exceed the capacity of e. In the restricted setting with unit bandwidth requirements and unit edge capacities, this means that no edge can be used by more than one call at the same time. The length of a schedule (the *makespan*) is the latest finishing time of all calls. Our goal is to minimize the makespan.

1.2 Our Results

In sections 2 and 3 we restrict ourselves to calls with unit durations and unit bandwidth requirements. We examine the call scheduling problem in two different variants of the general model given in the preceding section. First, in section 2, we consider bidirectional calls. Here, each call requires the establishment of two connections, one from the source node to the target node and one in the opposite direction, both having the same bandwidth requirement. In practice, such connections occur in the context of telephone calls or video conferences. If we require, in addition, that both connections associated with a call use the same (undirected) path in the network, we can simplify the model and represent the network by an undirected graph. We show that the call scheduling problem can be solved in polynomial time if G is a binary tree. For arbitrary trees, however, we prove the problem to be $\mathcal{NP}$–hard. Using a result by Nishizeki and Kashiwagi,[13] we give an (asymptotic) 1.1–approximation algorithm for this case. Furthermore, we prove that the problem is $\mathcal{NP}$–hard for ring networks as well. A 2–approximation algorithm can be specified easily, but better approximation seems difficult.

In section 3 we examine the case where calls are directed from a source node to a target node, without any need for a connection in the opposite direction. Such connections are used for data transfer or movie transmission, for example. Again, we show that the call scheduling problem is $\mathcal{NP}$–hard for trees and rings.

Finally, in section 4, we present $\mathcal{NP}$–hardness results for scheduling of calls with arbitrary bandwidth requirements on a single link, for scheduling of calls with arbitrary durations in line networks, and for scheduling of calls with either arbitrary bandwidth requirements or arbitrary durations in graphs that contain at least two edge-disjoint paths between some pair of nodes.

1.3 Related Work

The on-line version of the call scheduling problem was examined by Feldmann et al.[5,6] They considered line networks and binary tree networks, assuming that all edges have the same capacity and that all calls are bidirectional. They showed that the greedy algorithm is $O(\log n)$–competitive for binary trees with n nodes, as long as bandwidth requirements are either not too big or not too small. In addition, they presented a non-greedy algorithm for line networks that is c–competitive for bandwidth requirements between $1/c$ and $1/2$.

The call scheduling problem considered in this paper is closely related to the wavelength routing problem in all-optical networks. The undirected version of that problem was examined by Raghavan and Upfal.[15] They derived results that are equivalent to some of the results we present in section 2. Their approximation algorithm for tree graphs can be improved using the results by Nishizeki and Kashiwagi[13] in the way we describe in this paper. The directed version of the wavelength routing problem was considered by Mihail, Kaklamanis and Rao.[12] They presented approximation algorithms for trees, rings and trees of rings. Our proof of $\mathcal{NP}$–hardness for the problem of directed call scheduling in trees and rings resolves the open question they mentioned.

Scheduling undirected calls in a tree is equivalent to coloring an edge intersection graph of paths in a tree. Such EPT graphs were examined by Golumbic and Jamison.[9,10] They showed that the recognition problem and the coloring problem are both $\mathcal{NP}$–complete for these graphs. Tarjan gave a $\frac{3}{2}$–approximation algorithm for coloring EPT graphs.[17]

2 Bidirectional Calls

If all calls are bidirectional and require the full bandwidth in both directions, we take $G = (V, E)$ to be an undirected graph and we assign each call a starting time and an undirected path in G such that no two calls that are active at the same time use the same edge. Besides, we assume that each call has unit duration. In line networks, this call scheduling problem is equivalent to the coloring problem for interval graphs, and therefore solvable in polynomial time.[16] In tree networks, call scheduling is more difficult.

2.1 Binary Tree Networks

We examine the case where $G = (V, E)$ is a binary tree with n nodes. Feldmann et al.[5] showed that in this setting the greedy algorithm can produce a schedule that is longer than the optimal schedule by a factor of $\Omega(\log n)$. We present a simple polynomial time algorithm that always computes a schedule with minimum makespan.

Let w be the root of the binary tree G (any node of G can be used as the root). We assign each node $v \in V$ a level according to the length of the path from w to v. The root has level 0. We denote the least common ancestor of two nodes u and v by $\mathrm{lca}(u, v)$. For a request $r = (u, v, b, d)$, we let z_r stand for the node $\mathrm{lca}(u, v)$. Our algorithm is as follows:

(1) assign to each node $v \in V$ the set $R_v \subseteq R$ of all requests $r \in R$ with $z_r = v$; call a request $r \in R_v$ a *two-sided* request if it uses both downward links from v, otherwise a *one-sided* request (more specific, a *left* request or a *right* request)

(2) **for** time steps $t = 0, 1, \ldots$ **do**

begin

{ compute the set $R^t \subseteq R$ of requests that are to be scheduled during time step t }

(3) **for** all nodes $v \in V$, processed in order of non-decreasing level **do**

begin

(4) choose 0, 1 or 2 requests in R_v that can be scheduled during time step t, according to the following rule:

1. if R_v contains a two-sided request r and both downward links of v are still available in this time step, choose that request r
2. otherwise, choose one left request r_1, if possible, and one right request r_2, if possible

(5) remove the chosen requests from R_v and schedule them now (i.e., include them in R^t)

end

end

(6) the algorithm terminates when all requests have been scheduled

Note that whenever the algorithm reaches step (4), at most one of the two downward links is reserved by a call that has been scheduled at a previous node. We call this algorithm the *level algorithm*. Obviously, it can be implemented to run in time polynomial in the size of V and R.

Theorem 2.1 *The schedule computed by the level algorithm is optimal.*

Proof: We describe how to transform any optimal schedule into the schedule produced by the level algorithm without increasing the makespan, thereby showing that the level algorithm is optimal. Let R be the set of call requests that is the input to the algorithm. Denote by t^* and t_L the makespan of the optimal schedule S^* and of the schedule S produced by the level algorithm, respectively. The transformation proceeds time step by time step.

Assume that S and S^* are identical for all time steps from 0 up to $t-1$. (This assumption holds trivially for $t = 0$.) We explain how to modify

S^* such that S and S^* become identical for all time steps from 0 up to t. This modification does not increase the makespan of S^*. Hence, S^* can be transformed into S by repeated application of such modifications, and the theorem follows.

For the modification of S^* at time step t, the nodes of G are processed in order of non-decreasing levels. When processing a node v at level l, we can therefore assume that S and S^* are already identical with respect to requests that pass through (or terminate at) nodes with level $< l$. We call these requests *fixed* requests (with respect to v). Note that there can be at most one fixed request (w.r.t. v) passing through v. Hence, at most one of the downward links leaving v can be in use by such a fixed request, and it is the same downward link both in S and in S^*. The remaining one or two downward links (*free* links) have not been assigned to fixed requests, but may be used by requests from R_v that are scheduled at this time step. We distinguish three cases in our presentation of the modifications that are necessary to make S and S^* identical w.r.t. node v at time step t (we call S and S^* *identical w.r.t. node v at time step t* if the requests scheduled on all edges incident to v during this time step are the same in S and S^*):

1. If there are no requests scheduled on the free downward links at time step t in schedule S, all requests of R_v that could be scheduled on these links have already been scheduled in previous time steps. Hence, S and S^* are already identical w.r.t. node v, and we can proceed to the next node.

2. If there is a two-sided request r of R_v scheduled at time step t in S, there must exist a time step $t' \geq t$ such that r is scheduled at time step t' in S^*. If $t \neq t'$, we exchange the requests scheduled in the subtree rooted at v in S^* at time step t and time step t'. Obviously, S^* remains a feasible schedule with the same makespan. S and S^* are now identical w.r.t. node v at time step t, and we can proceed to the next node.

3. If there are one or two one-sided requests r_1 and (possibly) r_2 scheduled at time step t in S, there must exist time steps $t'_1 \geq t$ and $t'_2 \geq t$ such that r_1 and r_2 are scheduled at t'_1 and t'_2 in S^*. If $t_1 \neq t'_1$ or $t_2 \neq t'_2$, we exchange the requests scheduled in the right or left subtrees of v in S^* at time steps t'_1 or t'_2 with the respective requests scheduled at time step t. This is always possible because it cannot occur that there is a two-sided request of R_v scheduled at t in S^* while there are two one-sided requests scheduled in S. Again, the exchanges preserve the feasibility of S^* and do not increase the makespan. S and S^* are now identical w.r.t. node v at time step t, and we can proceed to the next node.

The exchanges performed at a node v do not affect the work we have done at nodes previously processed at this time step. Therefore, after processing at most n nodes, schedules S and S^* are identical for time step t. ◻

2.2 Arbitrary Tree Networks

Now we deal with arbitrary tree networks, i.e. connected, acyclic, undirected graphs without limitations on the degree of a node. For a given tree network $G = (V, E)$ and a set R of connection requests, we denote by R_w (for any $w \in V$) the subset of R that contains all requests $(u, v, b, d) \in R$ with $w \in \mathrm{PATH}(u, v)$. Here, $\mathrm{PATH}(u, v)$ is the set of nodes on the (unique) path from u to v in G. Note that this definition of R_w is different from the definition in section 2.1.

Theorem 2.2 *For any set R of requests, the minimum makespan* $\mathrm{OPT}(R)$ *is given by* $\mathrm{OPT}(R) = \max_{w \in V} \{\mathrm{OPT}(R_w)\}$.

Proof: Since $R_w \subseteq R$ for all $w \in V$, we have $\mathrm{OPT}(R) \geq \max_{w \in V} \{\mathrm{OPT}(R_w)\}$. All we need to do is show that it is indeed possible to construct a schedule of length $\max_{w \in V} \{\mathrm{OPT}(R_w)\}$. Assume that we are given optimal schedules $S(R_w)$ of length $\mathrm{OPT}(R_w)$ for all $w \in V$. We show how to merge them into an optimal schedule $S(R)$ without using more time steps than needed for that set R_w with the greatest schedule length.

The merging process starts at an arbitrary node v. Initially, we set $S := S(R_v)$ and mark v as the only node that has already been processed. During the whole construction the length of S will remain less than or equal to $\max_{w \in V} \{\mathrm{OPT}(R_w)\}$. We repeatedly merge S and a schedule $S(R_w)$ where w is a node that has not been processed yet but that is adjacent to a previously processed node. Note that the node w is adjacent to exactly one previously processed node u, and that all calls that are contained in both S and $S(R_w)$ use the edge $\{u, w\}$. Call these calls the *intersecting* calls. Let T denote the subtree of G that contains w and all nodes reachable from w without using edge $\{u, w\}$. The intersecting calls are the only calls in S that touch T. Furthermore, the intersecting calls are the only calls in $S(R_w)$ that touch $G \setminus T$. Hence, the intersecting calls are the only calls that have to be taken care of when merging S and $S(R_w)$.

We permute the schedule $S(R_w)$ in such a way that the resulting schedule is a feasible schedule for R_w with the intersecting calls being scheduled at the same time steps as in S (we may have to introduce idle time steps for that purpose). It is possible to permute $S(R_w)$ because all calls have unit duration: one can simply exchange all the calls scheduled at time step t with the calls scheduled at time step t'. Since the permuted schedule for R_w and the schedule

S are "compatible" (the intersecting calls are scheduled at the same time step), we can merge them in the obvious way and obtain a schedule S'. Neither the length of the permuted schedule for R_w nor the length of S' are greater than $\max\{\text{length}(S), \text{OPT}(R_w)\}$. Therefore, we can use S' as the new S. We mark w as processed and continue with the next node. This process terminates after finitely many steps. The schedule S we obtain in the end has length not greater than $\max_{w \in V}\{\text{OPT}(R_w)\}$. Hence, it is an optimal schedule $S(R)$ for the whole set of calls R. ◻

This result suggests an algorithm for solving the call scheduling problem in arbitrary tree networks: solve the problem independently for all sets R_w, and then merge the solutions according to the algorithm outlined in the proof of the theorem. Unfortunately, the following theorem shows that the problem of call scheduling for a single set R_w is already very hard.

Theorem 2.3 *It is $\mathcal{NP}$–hard to compute* $\text{OPT}(R_w)$ *in arbitrary tree networks.*

Proof: (by reduction from EDGE-COLORING)
We outline a polynomial time transformation of an instance of the EDGE-COLORING problem into an instance of the call scheduling problem with all calls passing through a single node. An instance of the EDGE-COLORING problem is a graph $G = (V, E)$ with maximum degree Δ. The question is whether the edges of G can be colored with Δ colors such that edges are assigned different colors if they share an endpoint. This problem is $\mathcal{NP}$–complete even for 3–regular graphs,[11] i.e. for regular graphs with $\Delta = 3$. The input for the scheduling algorithm is a network $G' = (V', E')$ with $V' = V \cup \{w\}$ (assuming w.l.o.g. that $w \notin V$) and $E' = \{\{w, u\} | u \in V\}$, and the set of call requests $R = \{(u, v, 1, 1) | \{u, v\} \in E\}$. The capacities $c(e)$ of all edges $e \in E'$ are 1. G' is a star-like graph and a special case of a tree network. Obviously, $R = R_w$. It is easy to see that every schedule for R_w in G' corresponds to a proper edge coloring of G: scheduling $(u, v, 1, 1)$ at time step t corresponds to coloring edge $\{u, v\}$ with color t. In particular, a minimum makespan schedule for R_w corresponds to an edge-coloring of G that uses the minimum number of colors. Thus, computing an optimal schedule for R_w yields an answer to the $\mathcal{NP}$–complete problem EDGE-COLORING, and therefore the problem to compute an optimal schedule for R_w is $\mathcal{NP}$–hard in general. ◻

Exploiting the relationship between call scheduling and edge-coloring further, we use a result by Nishizeki and Kashiwagi[13] to devise an approximation algorithm for the call scheduling problem.

Theorem 2.4 *There is a polynomial time approximation algorithm for the call scheduling problem in arbitrary tree networks, yielding a schedule of length no greater than* $\lfloor 1.1 \cdot \text{OPT}(R) + 0.8 \rfloor$.

Proof: Let $G = (V, E)$ be the given tree network. We show how to construct a schedule $S(R_w)$ for each set R_w, $w \in V$, satisfying $\text{length}(S(R_w)) \leq \lfloor 1.1 \cdot \text{OPT}(R_w) + 0.8 \rfloor$. Using the merging algorithm outlined in the proof of Theorem 2.2, it is obvious that we can then compute a schedule for R of length $\max_{w \in V} \{\text{length}(S(R_w))\} \leq \max_{w \in V} \{\lfloor 1.1 \cdot \text{OPT}(R_w) + 0.8 \rfloor\} = \lfloor 1.1 \cdot \text{OPT}(R) + 0.8 \rfloor$.

In order to solve the scheduling problem for a single set R_w, we proceed as follows. Let $w_1, w_2, \ldots, w_k$ be the neighbours of w in G. We construct a graph $G' = (V', E')$ with $V' = \{1, 2, \ldots, k\}$. Each call in R_w is translated into an edge of G'. Calls starting or ending at w and touching node w_i contribute a self-loop (i, i) to E'. Calls touching node w and two neighbours w_i and w_j contribute an edge (i, j) to E'. Hence, G' is a multigraph with self-loops, and each proper edge-coloring of G' corresponds to a feasible schedule for R_w. To see this, note that two calls in R_w are in conflict if and only if both of them touch a neighbour w_i of w, which is the case if and only if the corresponding edges in G' share an endpoint. In particular, an optimal schedule for R_w corresponds to an optimal edge-coloring of G'. Using the algorithm by Nishizeki and Kashiwagi,[13] one can compute an edge-coloring for G' that uses no more than $\lfloor 1.1 \cdot \chi'(G') + 0.8 \rfloor$ colors, where $\chi'(G')$ is the chromatic index of G'. The algorithm was originally presented for multigraphs without self-loops, but it is easy to see that self-loops do not cause any problems. One can temporarily remove the self-loops from G', run the edge-coloring algorithm on this reduced graph, and then color the self-loops afterwards with available colors in a greedy manner. This does not increase the ratio between the number of colors used by the algorithm and the optimal number of colors.

Having computed the near–optimal edge-coloring of G', one can translate this coloring into a near–optimal schedule for R_w: different colors simply correspond to different time steps. The approximation ratio remains the same after this translation. Hence, we have computed a schedule for R_w of length no greater than $\lfloor 1.1 \cdot \text{OPT}(R_w) + 0.8 \rfloor$. ◻

2.3 Ring Networks

The call scheduling problem in ring networks is related to the problem of coloring circular-arc graphs. A graph $G = (V, E)$ is a circular-arc graph if its vertices can be represented by arcs of a circle such that there is an edge between two vertices in G if and only if the corresponding arcs intersect. The vertex-coloring problem for circular-arc graphs (ARC-COLORING) is $\mathcal{NP}$–complete.[8] If each call in a call scheduling problem for rings had to specify which of the two alternative routes in the ring it takes, the scheduling prob-

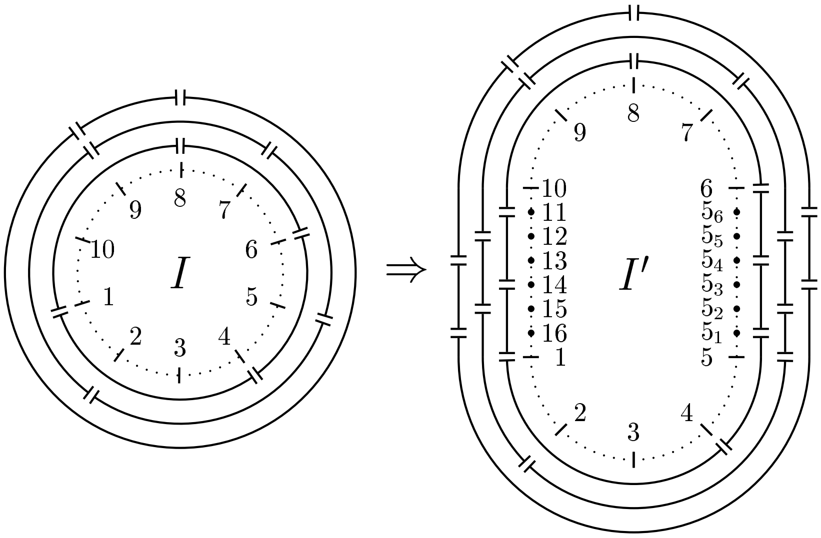

Figure 1: Instances I and I' of the ARC-COLORING problem

lem would be equivalent to ARC-COLORING. Since we have the additional freedom to choose one of two possible routes for each call, however, it is not immediately clear whether the call scheduling problem for ring networks is $\mathcal{NP}$–hard. Nevertheless, we can establish this fact with a transformation from ARC-COLORING.

Theorem 2.5 *The call scheduling problem in ring networks is $\mathcal{NP}$–hard.*

Proof: (by reduction from ARC-COLORING)
Let an instance I of the ARC-COLORING problem be given by a family $F = \{A_1, \ldots, A_n\}$ of circular arcs and a positive integer K. The problem is to decide whether F can be colored with K colors such that intersecting arcs are assigned different colors.

Each arc $A_i \in F$ is a pair (a_i, b_i) with $a_i, b_i \in \{1, \ldots, m\}$. We call a_i and b_i the *endpoints* of arc A_i. The *span* $\mathrm{sp}(A_i)$ of arc A_i is the set $\{a_i+1, a_i+2, \ldots, b_i\}$ if $a_i < b_i$ or $\{a_i+1, \ldots, m, 1, 2, \ldots, b_i\}$ if $a_i > b_i$. Two arcs A_i and A_j intersect if $\mathrm{sp}(A_i) \cap \mathrm{sp}(A_j) \neq \emptyset$. Without loss of generality, we can assume that $m \leq 2n$ and that each point i, $1 \leq i \leq m$, is contained in the span of exactly K arcs. (For the latter, note that one can simply add arcs of the form $(i, i+1)$ or $(m, 1)$ until the condition is satisfied.)

First, we transform I into an instance I' of the ARC-COLORING problem that can be colored with K colors if and only if I can be colored with K colors. Whereas I can contain arcs that span more than half of the circle, the length of all arcs in I' will be strictly smaller than half the circumference of the (slightly larger) circle. Afterwards, we will prove that the instance I', if viewed as an instance I'' of the call scheduling problem, can be scheduled in K time steps if and only if I can be colored with K colors.

In order to transform I into I', we introduce $4K$ new points on the circle, $2K$ points between m and 1 and $2K$ points between $\lfloor m/2 \rfloor$ and $\lfloor m/2 \rfloor + 1$. We number the new points between m and 1 by $m+1, m+2, \ldots, m+2K$. By our assumption, there are exactly K arcs $B_1, \ldots, B_K$ spanning point 1. We replace each such arc $B_i = (u_i, v_i)$ by three new arcs $B_i^1 = (u_i, m+i)$, $B_i^2 = (m+i, m+K+i)$ and $B_i^3 = (m+K+i, v_i)$. Each of the points $m+1, \ldots, m+2K$ is an endpoint of exactly two arcs. Every two arcs that share such an endpoint must necessarily be assigned the same color in any proper K-coloring for the new family of arcs because they intersect the same $K-1$ other arcs. Hence, B_i^1, B_i^2 and B_i^3 must be assigned the same color in such a coloring, and any K-coloring for the modified instance corresponds to a K-coloring for I and vice versa.

The insertion of $2K$ new points between $\lfloor m/2 \rfloor$ and $\lfloor m/2 \rfloor + 1$ works the same way as between m and 1. Here, we replace each arc C_i from a set $\{C_1, \ldots, C_K\}$ by three new arcs C_i^1, C_i^2 and C_i^3. Having inserted $4K$ new points altogether, we obtain the instance I' of the ARC-COLORING problem. It is clear that the transformation can be done in polynomial time and that there is a K-coloring for I' if and only if there is a K-coloring for I. Figure 1 shows an instance I of the ARC-COLORING problem with $K = 3$ and the instance I' obtained from I by inserting $4K = 12$ new points.

Now we interpret I' as an instance I'' of the call scheduling problem. Each arc (a_i, b_i) is viewed as a call between nodes a_i and b_i on a ring network with $m + 4K$ nodes. We claim that there is a schedule for I'' of length no more than K if and only if there is a proper K-coloring for I'. First, it is possible to generate a schedule of length K for I'' from a K-coloring for I' by routing each call in the direction indicated by the corresponding arc and scheduling it at the time step given by the color of that arc. Second, a schedule of length K for I'' automatically gives a K-coloring for I'. This follows from the fact that all calls in I'' must be routed the short way in a schedule of length K, and the fact that the corresponding arc of a call corresponds to the short route for this call. (Note that every call in I'' uses more edges if routed the long way ($\geq m/2 + 2K + 1$ edges, if m is even, and $\geq (m+1)/2 + 2K$ edges, if m is odd) than it uses if routed the short way ($\leq m/2 + 2K - 1$ edges, if m is even, and $\leq (m-1)/2 + 2K$ edges, if m is odd). If each call is routed the short way in a schedule of length K, each edge of the ring is busy in every time step. Therefore, if at least one call is routed the long way, the schedule must have length greater than K.)

To obtain a K-coloring for I from a schedule of length K for I'', we simply take the K-coloring for I' which we get immediately and derive a K-coloring for I by coloring each arc of I with the color of all corresponding arcs in I'. (We already showed that these arcs are colored with the same color in any K-coloring for I'.) Note that, given a particular arc in I, we have up to five corresponding arcs in I', which are viewed as calls in I''. In any schedule of length K for I'', these calls must be scheduled at the same time step. ◻

For the wavelength routing problem in all-optical networks, there is a 2–approximation algorithm for rings that works both for the undirected[15] and the directed[12] case. We observe that this algorithm works for call scheduling as well.

Theorem 2.6 *There is a polynomial time approximation algorithm for the call scheduling problem in ring networks, yielding a schedule that is at most twice as long as an optimal schedule.*

Proof: The algorithm is as follows: remove some arbitrary edge from the ring and compute an optimal schedule S for the calls on the remaining line network. The latter step can be performed optimally since it is equivalent to the coloring problem in interval graphs.[16] We only need to show that an optimal schedule for the calls on the line network is at most twice as long as an optimal schedule on the ring.

Assume that we are given an optimal schedule S^* for the calls in the ring network, and that we have removed edge e from the ring in order to obtain the line network. Let ℓ^* be the length of schedule S^*. Then there are no more than ℓ^* calls using edge e in schedule S^*, at most one in each time step. If we remove these calls from S^* and schedule them in time steps after ℓ^*, one after the other, each routed such that it does not use edge e, we obtain a schedule S' of length at most $2\ell^*$ for the calls on the line network. Hence, the optimal schedule S for the calls on the line network has length at most $2\ell^*$. Therefore, S is a schedule of length at most $2\ell^*$ for the calls in the ring network. ◻

We want to note that the algorithm from the proof of this theorem is not a good choice for practical implementations. Other algorithms perform much better on the average. For example, an algorithm that starts each time step by scheduling the shortest remaining call and filling up the ring in a greedy manner performs close to optimal on most instances. Nevertheless, it is possible to devise worst-case inputs for which this algorithm has an approximation ratio arbitrarily near to 2. Up to our knowledge, no approximation algorithm with

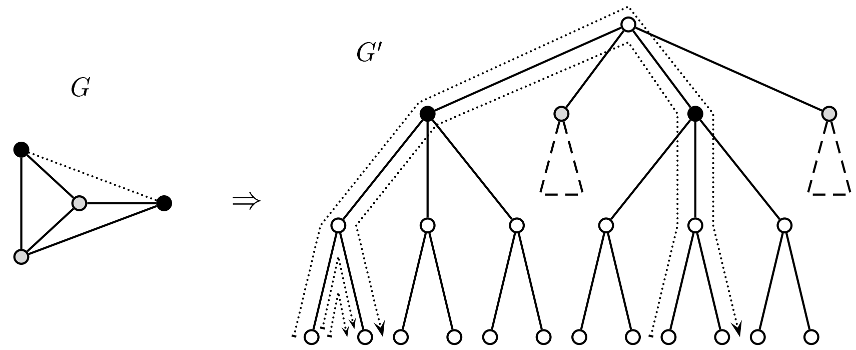

Figure 2: Reduction from EDGE-COLORING

an approximation ratio substantially better than 2 has been published for the call scheduling problem in rings. For the ARC-COLORING problem no such algorithm is known either, except for special classes of input graphs (see the work of Tucker[18] or Orlin et al.[14]).

3 Directed Calls

Having examined the case of bidirectional calls in section 2, we now turn our attention to the model with directed calls. First, we note that the problem is again solvable in polynomial time for line networks, since it is equivalent to the coloring of two independent interval graphs. In the following, we show that the scheduling problem for directed calls is $\mathcal{NP}$–hard for trees and rings.

Theorem 3.1 *Optimal scheduling of directed calls in tree networks of arbitrary degree is $\mathcal{NP}$–hard.*

Proof: (by reduction from EDGE-COLORING)
The basic idea of the transformation is the same as in the proof of Theorem 2.3, but directed calls require a more involved construction. Let $G = (V, E)$ be a regular graph with degree 3. It is $\mathcal{NP}$–complete to decide whether G can be properly edge-colored with 3 colors.[11] We show how to transform any 3–regular graph G into an instance of the scheduling problem for directed calls in a tree network $G' = (V', E')$. Here, V' contains all nodes $v \in V$ and 9 additional nodes $v_1, v_2, v_3, v_{11}, v_{12}, v_{21}, v_{22}, v_{31}, v_{32}$ for each $v \in V$, as well as a new node w. Formally, $V' = \{w\} \cup \{v, v_1, v_2, v_3, v_{11}, v_{12}, v_{21}, v_{22}, v_{31}, v_{32} | v \in V\}$. The edges in E' are such that w is the root of G', $v \in V$ are the children of w, and v_1, v_2, v_3 are the children of v, and v_{i1}, v_{i2} are the children of v_i. Formally,

$E' = \{\{w, v\}, \{v, v_i\}, \{v_i, v_{ij}\} | v \in V, i \in \{1, 2, 3\}, j \in \{1, 2\}\}$.

The set R of requests contains four calls for each edge $e = \{u, v\} \in E$, namely $r_1^e = (u_{i1}, v_{j2}, 1, 1)$, $r_2^e = (v_{j1}, u_{i2}, 1, 1)$, and $r_3^e = r_4^e = (u_{i1}, u_{i2}, 1, 1)$. The values of i and j are selected from $\{1, 2, 3\}$ such that a different value of i [of j] is chosen for each edge incident to u [incident to v]. Intuitively, r_1^e and r_2^e correspond to the single bidirectional call we used in the proof of Theorem 2.3, and the other two calls r_3^e and r_4^e are merely blockers that make sure that r_1^e and r_2^e are scheduled at the same time step in any schedule of length 3. A 3–regular graph and part of the resulting instance of the call scheduling problem are sketched in Figure 2. The nodes of the graph G on the left side correspond to the children of the root of the graph G' on the right side. The black nodes in G correspond to the black children of the root in G'. The dotted edge between the black nodes in G corresponds to the four dotted calls indicated in G'. The subtrees rooted at the children of the root of G' are shown only for the two relevant nodes.

Now we show that there is a schedule of length 3 for R in G' if and only if a proper edge-coloring with 3 colors exists for G. Assume that we have a 3-coloring of G. For each edge $e \in E$, we schedule the calls r_1^e and r_2^e in the time step given by the color of edge e. The calls r_3^e and r_4^e are scheduled in the two time steps that are still available. Obviously, this yields a valid schedule of length 3 for R in G'.

In the other direction, assume that there is a schedule of length 3 for R in G'. The blockers make sure that in such a schedule r_1^e and r_2^e (for each $e \in E$) are assigned the same time step. No call r_1^f with $e \neq f \in E$ and $f \cap e \neq \emptyset$ can be scheduled at this time step. As a consequence, if we give each edge $e \in E$ the color corresponding to the time step assigned to the call r_1^e, we obtain a proper 3-coloring for the edges of G.

We have shown that computing an optimal schedule for R yields an answer to the $\mathcal{NP}$–complete problem EDGE-COLORING. Since the construction of G' and R can be done in polynomial time, the theorem follows. ◻

Considering the results for bidirectional calls in section 2, one might expect that the scheduling problem for directed calls is easier in binary trees, too. In another paper,[4] however, we show by a reduction from ARC-COLORING that the scheduling problem for directed calls is $\mathcal{NP}$–hard already for binary trees.

For directed calls in ring networks, we obtain results that are the same as those obtained for bidirectional calls in section 2.3.

Theorem 3.2 *Optimal scheduling of directed calls in ring networks is $\mathcal{NP}$–hard.*

Proof: (by reduction from ARC-COLORING)
To prove this result, we adapt the proof of Theorem 2.5 to directed calls. Let

I be an instance of the ARC-COLORING problem. We transform it into an equivalent instance I' of the ARC-COLORING problem, just as we did in the proof for bidirectional calls. Now we view I' as an instance I'' of the call scheduling problem for directed calls, substituting two calls (u, v) and (v, u) for each arc (u, v) of I'.

In any schedule of length K for I'', all calls must be routed the short way. (This follows from the same argument as presented in the proof of Theorem 2.5.) Such a schedule actually consists of two independent schedules, one for the calls routed in clockwise direction and one for the calls routed in counter-clockwise direction, each in a directed ring with the corresponding orientation. Each of these two schedules contains exactly one of the two calls corresponding to every arc of I', and can therefore be used to obtain a K-coloring for I'.

Similarly, any K-coloring for I' can be used to obtain a schedule of length K for I'', scheduling both calls corresponding to an arc at the time step given by the color assigned to that arc in the K-coloring of I', and routing all calls the short way. This concludes the proof of the theorem. □

We remark that a 2–approximation algorithm for scheduling directed calls in ring networks can be obtained the same way as for scheduling bidirectional calls. Removing an arbitrary edge from the given ring leads to two separate call scheduling problems in independent line networks. These can be solved optimally in polynomial time using interval coloring algorithms. The resulting schedule for the ring network is at most twice as long as an optimal schedule. This fact was observed by Mihail, Kaklamanis and Rao in the context of wavelength routing problems in all-optical networks.[12]

4 Arbitrary Durations and Bandwidths

Having examined the case of unit call durations and unit bandwidths in the previous sections, we consider the more general case without these restrictions now. We still assume that the bandwidth of all links in the network is the same, but now it can be any natural number $B \in \mathbb{N}$. A call can request any bandwidth $b \in \{1, 2, \ldots, B\}$. Similarly, the duration of a call can be any natural number.

4.1 Line Networks

Whereas call scheduling in line networks is solvable in polynomial time in the case of unit bandwidth requirements and call durations, it becomes equivalent to the most general one-dimensional layout compaction problem[3] once we allow arbitrary call durations. This equivalence was pointed out by Anja Feldmann.[6]

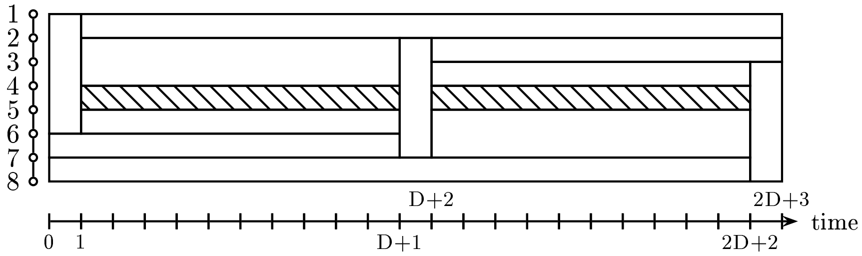

Figure 3: Minimum makespan schedule for the additional calls

The proof of the following theorem is a modification of the corresponding proof by Doenhardt and Lengauer.[3] Whereas they present a reduction from 3-PARTITION and thereby prove their decision problem $\mathcal{NP}$–complete in the strong sense, we choose to give a simpler proof employing a reduction from PARTITION. Our proof works already for line networks with 8 nodes while the reduction from 3-PARTITION requires line networks of arbitrary size.

Theorem 4.1 *Scheduling of calls with unit bandwidth requirements and arbitrary durations is $\mathcal{NP}$–hard for any line network with ≥ 8 nodes and unit edge capacities. This result holds for bidirectional calls as well as for directed calls.*

Proof: (by reduction from PARTITION)
In line networks the call scheduling problem for directed calls can be solved by solving two independent instances of the call scheduling problem for bidirectional calls, one corresponding to all calls from left to right and one to all calls in the opposite direction. Thus, it is enough to show $\mathcal{NP}$–hardness for the scheduling problem with bidirectional calls.

An instance of the $\mathcal{NP}$–complete PARTITION problem[7] is given by a finite set A with size $s(a) \in \mathbb{N}$ for each $a \in A$ such that $\sum_{a\in A} s(a)$ is even. The question is whether there is a subset $A' \subseteq A$ such that $\sum_{a\in A'} s(a) = \sum_{a\in A\setminus A'} s(a)$.

Let $D = \left(\sum_{a\in A} s(a)\right)/2$. Let $G = (V, E)$ be a line network with at least 8 nodes. Select any 8 consecutive nodes of G and number them 1 through 8. The request set R that will be the input for the call scheduling algorithm contains one request $(4, 5, s(a))$ for each $a \in A$, and 11 additional requests $(1, 6, 1), (1, 2, 2D+2), (2, 4, D), (2, 7, 1), (2, 3, D+1), (3, 4, D), (3, 8, 1), (5, 6, D)$, $(5, 7, D), (6, 7, D+1)$ and $(7, 8, 2D+2)$. (A tuple (u, v, d) denotes a bidirectional request between nodes u and v of duration d.) In the following, we show that there is a schedule of length $2D + 3$ for R if and only if A can be partitioned into two subsets of equal size.

First, consider only the 11 additional requests. Obviously, they cannot be scheduled in less than $2D+3$ time steps, since edge $(1,2)$ (and five other edges) of the line network needs to service calls whose durations add up to $2D+3$. Figure 3 shows one possible schedule of length $2D+3$ for the additional calls. It is easy to see that the only other schedule of minimum makespan is the reverse of the depicted schedule with respect to the time axis.

In any of the two possible minimum makespan schedules for the additional calls, there are two separate time intervals of length D during which the edge $(4,5)$ is available. (In Figure 3, these time intervals are drawn shaded.) The remaining calls, all of which have the form $(4,5,s(a))$, can be scheduled in these two separate time intervals if and only if A can be partitioned into two subsets of size D. $\square$

The additional call requests used in this proof force two separate gaps of equal length in an optimal schedule. Allowing a larger number of nodes in the line network, one can have additional call requests that force an arbitrary number of separate gaps in an optimal schedule. This can be exploited to prove that the call scheduling problem for arbitrary line networks is $\mathcal{NP}$–complete in the strong sense. Details can be found in the article by Doenhardt and Lengauer.[3]

We have seen that, as soon as we allow arbitrary call durations, the call scheduling problem becomes $\mathcal{NP}$–hard for line networks with ≥ 8 nodes. This result implies $\mathcal{NP}$–hardness for scheduling calls with arbitrary duration in all tree networks with diameter ≥ 8.

Next, we examine the case that we have unit call durations but allow arbitrary bandwidth requirements.

Theorem 4.2 *If calls have unit duration and arbitrary bandwidth requirements, optimal call scheduling is $\mathcal{NP}$–hard even for a line network consisting of a single edge.*

Proof: (by reduction from PARTITION)
Let an instance of the PARTITION problem be given by a finite set A with size $s(a) \in \mathbb{N}$ for each $a \in A$, and let $B = \left(\sum_{a \in A} s(a)\right)/2$. The input for the call scheduling algorithm is a network $G = (V,E)$ that consists of a single link (u,v) with capacity B, and a set of call requests R that contains one call $(u,v,s(a),1)$ for each $a \in A$. There is a schedule of length 2 for R if and only if A can be partitioned into two subsets of equal size. $\square$

We remark that this result implies $\mathcal{NP}$–hardness for any line or tree network. In addition, it is clear from the proof that there cannot be a polynomial time approximation algorithm with absolute approximation ratio less than 3/2 for call scheduling with arbitrary bandwidths in these networks unless $\mathcal{P} = \mathcal{NP}$.

4.2 Networks with Alternative Routes

Next, we examine networks that contain two nodes u and v with at least two edge-disjoint simple paths connecting u and v. Our results apply to both bidirectional and directed calls. Note that the maximum number of edge-disjoint paths between two nodes can be computed in polynomial time with a maximum flow algorithm.[1]

Theorem 4.3 *Let $G = (V, E)$ be a network with ≥ 2 edge-disjoint simple paths between u and v (for some $u, v \in V$). If calls have unit bandwidth requirements and arbitrary durations, optimal call scheduling is $\mathcal{NP}$–hard.*

Sketch of Proof: (by reduction from PARTITION)
Let an instance of the PARTITION problem be given by a finite set A with size $s(a) \in \mathbb{N}$ for each $a \in A$, and let $D = \left(\sum_{a\in A} s(a)\right)/2$. Let k be the maximum number of edge-disjoint paths between u and v in G. The input for the call scheduling algorithm is the set of call requests R that contains one call $(u, v, 1, s(a))$ for each $a \in A$ and $k-2$ additional calls $(u, v, 1, D)$. There is a schedule of length D for R if and only if A can be partitioned into two subsets of equal size. ◻

Theorem 4.4 *Let $G = (V, E)$ be a network with ≥ 2 edge-disjoint simple paths between u and v (for some $u, v \in V$). If calls have arbitrary bandwidth requirements and unit durations, optimal call scheduling is $\mathcal{NP}$–hard.*

Sketch of Proof: (by reduction from PARTITION)
Let an instance of the PARTITION problem be given by a finite set A with size $s(a) \in \mathbb{N}$ for each $a \in A$, and let $B = \left(\sum_{a\in A} s(a)\right)/2$. Let k be the maximum number of edge-disjoint paths between u and v. The input for the call scheduling algorithm is the graph G with edge capacities B and the set of call requests R that contains one call $(u, v, s(a), 1)$ for each $a \in A$ and $k-2$ additional calls $(u, v, B, 1)$. There is a schedule of length 1 for R if and only if A can be partitioned into two subsets of equal size. ◻

A consequence of the proof is that there cannot be a polynomial time approximation algorithm with absolute approximation ratio less than 2 for call scheduling with arbitrary bandwidths unless $\mathcal{P} = \mathcal{NP}$.

5 Concluding Remarks

We have considered the off-line version of the call scheduling problem in various settings. Our results have shown many variants of call scheduling $\mathcal{NP}$–hard, even in very restricted settings. We have given a polynomial time optimal algorithm for bidirectional calls in binary trees, an approximation algorithm for bidirectional calls in arbitrary trees, and approximation algorithms for both bidirectional and directed calls in ring networks. All algorithms assume that the calls have unit bandwidth requirements and unit durations. This restricted problem is equivalent to the wavelength routing problem in all-optical networks, and therefore of practical interest. It is an interesting question to investigate this call scheduling problem for other network topologies, for example for mesh networks.

In ATM networks, connection requests can specify arbitrary bandwidth requirements and calls can have arbitrary durations. Hence, there is a need for good approximation algorithms in this general setting. To our knowledge, no such approximation algorithms have been published so far.

There are some results by Feldmann et al.[5,6] for on-line call scheduling. In an on-line scenario the scheduling algorithm has no information about call durations or future connection requests. Since many call scheduling problems that will arise in real applications are likely to be on-line problems, the development of competitive on-line algorithms is a very promising direction of research.

Acknowledgments

This work was partly funded by the German Science Foundation (DFG), Contract: SFB 342, TP A7.

References

1. R.K. Ahuja, T.L. Magnanti, and J.B. Orlin: *Network Flows: Theory, Algorithms, and Applications.* Prentice Hall, New Jersey, 1993.
2. The ATM Forum. ATM User-Network Interface (UNI) Specification Version 3.1. Prentice Hall PTR, Upper Saddle River, NJ, 1995.
3. J. Doenhardt and T. Lengauer. Algorithmic aspects of one-dimensional layout compaction. *IEEE Transactions on Computer-Aided Design*, CAD-6(5):863–878, September 1987.
4. T. Erlebach and K. Jansen. Call scheduling in trees and rings. unpublished manuscript, 1996.
5. A. Feldmann, B. Maggs, J. Sgall, D. Sleator, and A. Tomkins. Competitive analysis of call admission algorithms that allow delay. Technical Report CMU-CS-95-102, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA, January 1995.
6. A. Feldmann. On-line call admission for high-speed networks (Ph.D. Thesis). Technical Report CMU-CS-95-201, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA, October 1995.

7. M.R. Garey and D.S. Johnson. *Computers and intractability. A guide to the theory of $\mathcal{NP}$-completeness.* W.H. Freeman and Company, New York-San Francisco, 1979.
8. M.R. Garey, D.S. Johnson, G.L. Miller, and C.H. Papadimitriou. The complexity of coloring circular arcs and chords. *SIAM J. Algebraic Discrete Methods*, 1(2):216–227, 1980.
9. M.C. Golumbic and R.E. Jamison. Edge and vertex intersection of paths in a tree. *Discrete Math.*, 55:151–159, 1985.
10. M.C. Golumbic and R.E. Jamison. The edge intersection graphs of paths in a tree. *J. Comb. Theory Series B*, 38(1):8–22, February 1985.
11. I. Holyer. The NP-completeness of edge-coloring. *SIAM J. Comput.*, 10(4):718–720, November 1981.
12. M. Mihail, C. Kaklamanis, and S. Rao. Efficient access to optical bandwidth. In *Proceedings of the 36th IEEE Symposium on Foundations of Computer Science*, pages 548–557, Los Alamitos-Washington-Brussels-Tokyo, 1995. IEEE, IEEE Computer Society Press.
13. T. Nishizeki and K. Kashiwagi. On the 1.1 edge-coloring of multigraphs. *SIAM J. Disc. Math.*, 3(3):391–410, August 1990.
14. J.B. Orlin, M.A. Bonuccelli, and D.P. Bovet. An $O(n^2)$ algorithm for coloring proper circular arc graphs. *SIAM J. Algebraic Discrete Methods*, 2(2):88–93, June 1981.
15. P. Raghavan and E. Upfal. Efficient routing in all-optical networks. In *Proceedings of the 26th Annual ACM Symposium on Theory of Computing STOC '94*, pages 134–143, New York, 1994. ACM SIGACT, ACM Press.
16. M. Ślusarek. A coloring algorithm for interval graphs. In A. Kreczmar and G. Mirkowska, editors, *Proceedings of the 14th Symposium on Mathematical Foundations of Computer Science (Porabka-Kozubnik, Poland, August/September 1989)*, LNCS 379, pages 471–480, Berlin-Heidelberg-New York-London-Paris-Tokyo-Hong Kong, 1989. Springer-Verlag.
17. R.E. Tarjan. Decomposition by clique separators. *Discrete Math.*, 55:221–232, 1985.
18. A. Tucker. Coloring a family of circular arcs. *SIAM J. Appl. Math.*, 29(3):493–502, November 1975.