Off-Line and On-Line Call-Scheduling in Stars and Trees*

Thomas Erlebach[1] and Klaus Jansen[2]

[1] Institut für Informatik, TU München, D–80290 München,
erlebach@informatik.tu-muenchen.de
[2] Fachbereich IV – Mathematik, Universität Trier, Postfach 3825, D–54286 Trier,
jansen@dm3.uni-trier.de

Abstract. Given a communication network and a set of call requests, the goal is to find a minimum makespan schedule for the calls such that the sum of the bandwidth requirements of simultaneously active calls using the same link does not exceed the capacity of that link. In this paper the call-scheduling problem is studied for star and tree networks. Lower and upper bounds on the worst-case performance of List-Scheduling (LS) and variants of it are obtained for call-scheduling with arbitrary bandwidth requirements and either unit call durations or arbitrary call durations. LS does not require advance knowledge of call durations and, hence, is an on-line algorithm. It has performance ratio (competitive ratio) at most 5 in star networks. A variant of LS for calls with unit durations is shown to have performance ratio at most $2\frac{2}{3}$. In tree networks with n nodes, a variant of LS for calls with unit durations has performance ratio at most 6, and a variant for calls with arbitrary durations has performance ratio at most $5 \log n$.

1 Introduction

Call-scheduling problems arise naturally in modern communication networks, e.g., ATM networks. ATM (*asynchronous transfer mode*) is a network protocol that allows high-bandwidth connections with a guaranteed quality of service [15]. A connection request (call) can specify a certain bandwidth requirement, and the network guarantees that, once the connection is established, this bandwidth is available to it as long as it remains active. Consequently, this bandwidth must be reserved on all links along a path that connects the endpoints of the call in the network. The high bandwidth and guaranteed quality of service in ATM networks are essential for upcoming applications like multimedia servers or real-time medical imaging.

Formally, the communication network is given by a connected, undirected graph $G = (V, E)$ such that each edge $e \in E$ has a certain capacity $c(e)$. We assume that all edges have the same capacity, and that this capacity is normalized to 1. A call request r is a tuple (u_r, v_r, b_r, d_r), where u_r and v_r are different nodes

* Partly supported by German Science Foundation (DFG), Contract: SFB 342 TP A7.

of G representing the endpoints of the connection, $b_r \in]0;1]$ is the requested bandwidth, and $d_r \in \mathbb{N}$ is the duration of the call. Given a graph G and a (multi-)set R of call requests, a feasible schedule S assigns to each request $r \in R$ a starting time $t_r \in \mathbb{N}_0$ and an undirected path P_r from u_r to v_r in G such that the sum of bandwidths of simultaneously active calls using the same edge does not exceed the capacity of that edge. Precisely speaking, call r is active during the time interval $[t_r; t_r + d_r[$, and it occupies bandwidth b_r on all edges of P_r during that time. Several active calls can share an edge if the sum of their bandwidth requirements is at most 1.

The length $|S|$ of a schedule S is the latest finishing time of all calls, i.e., $|S| = \max_{r \in R} t_r + d_r$. We denote by $OPT = OPT(R)$ the length of a shortest feasible schedule for R, and by $A(R)$ the length of the schedule produced by algorithm A. Since it is in general $\mathcal{NP}$-hard to compute a minimum makespan schedule [5], one is interested in polynomial-time approximation algorithms with provable performance guarantee. An algorithm A has *performance ratio* at most ρ if $A(R) \leq \rho \cdot OPT(R)$ for all request sets R.

If an algorithm does not require advance knowledge of call durations, we refer to it as an *on-line* algorithm even if it requires that all call requests are given to the algorithm at once. Such *batch-style* on-line algorithms can easily be converted into *fully on-line* algorithms, i.e., algorithms that can deal with additional call requests that arrive on-line while other calls have already been scheduled, increasing the competitive ratio by no more than a factor 2 [14, 7]. An on-line algorithm has *competitive ratio* ρ if it always produces a schedule with makespan at most a factor ρ longer than the optimum (off-line) schedule.

If G is a tree, the path P_r is already completely determined by u_r and v_r. For an edge e of a tree network and a request set R we call $L(e) = \sum_{r \in R: e \in P_r} b_r \cdot d_r$ the *load* of edge e. Furthermore, L_{max} is the maximum of $L(e)$ over all $e \in E$. Obviously, L_{max} is a lower bound for the optimum schedule length. A special case of a tree is a graph that consists of a central node c and an arbitrary number of nodes $v_1, v_2, \ldots, v_k$ that are adjacent to c but not adjacent to each other. We refer to such graphs as *stars*. In the following, we will always assume that G is a star or a tree. Scheduling calls with unit bandwidth requirements in stars with unit edge capacities is equivalent to scheduling multiprocessor tasks with prespecified processor allocations if each task requests one or two processors [11].

One of the earliest heuristics for the solution of scheduling problems was List-Scheduling (LS), introduced by Graham [9]. In the call-scheduling context, the input to LS is a star or tree network G and a set R of call requests arranged in a list L. LS starts to schedule calls at time 0. If there is a call r in L such that bandwidth b_r is available on all edges along path P_r, LS schedules the first such call r in L and removes it from L; otherwise, it waits until one of the active calls finishes. This is repeated until all calls have been scheduled.

One important property of list-schedules is that, if a call request r is established at time t_r, it follows that at any time prior to t_r at least one of the edges on path P_r did not have bandwidth $\geq b_r$ available. We will use this property as a tool to prove performance guarantees for list-schedules. Note that this property holds only because there are no precedence constraints for the call requests.

Allowing arbitrary bandwidth requirements, we will show that $DBLS(L) \leq \frac{8}{3} OPT(L)$ for calls with unit durations in stars, that $LS(L) \leq 5 \cdot OPT(L)$ for calls with arbitrary durations in stars, that $LLS(L) \leq 6 \cdot OPT(L)$ for calls with unit durations in trees, and that $LSL(L) \leq 5 \log n \cdot OPT(L)$ for calls with arbitrary durations in trees with n nodes. DBLS, LLS, and LSL are variants of LS that will be defined later.

1.1 Related Work

Wavelength Allocation. The off-line call-scheduling problem with unit durations and unit bandwidth requirements is equivalent to the wavelength allocation problem in all-optical networks with wavelength-division multiplexing, where a minimum makespan schedule corresponds to a wavelength assignment with the minimum number of distinct wavelengths. Routing and wavelength allocation in all-optical networks have received considerable attention lately, see, e.g., [1] and the references contained in there.

A variation of the problem dealing with directed instead of bidirectional calls has also been studied. Here, calls using the same edge can receive the same wavelength if they use the edge in different directions. The best approximation algorithm known up to now requires $\frac{5}{3} L_{max}$ wavelengths in the worst case [12]. It is known that the bidirectional call-scheduling problem with unit durations and unit bandwidths is $\mathcal{NP}$-hard in trees of arbitrary degree, but solvable in polynomial time in trees whose degree is bounded by a constant [5, 6]. The directed version is $\mathcal{NP}$-hard already for binary trees [6].

In the on-line version of the wavelength allocation problem the algorithm is given requests one by one and must assign wavelengths immediately without knowledge about future requests. Bartal and Leonardi [3] obtain deterministic on-line algorithms with competitive ratio $O(\log n)$ for networks with n processors whose topology is that of a tree, a tree of rings, or a mesh. In addition, they present a matching lower bound of $\Omega(\log n)$ for all on-line algorithms for wavelength allocation in meshes, and a lower bound of $\Omega(\frac{\log n}{\log \log n})$ for trees. Note that the on-line version of the wavelength allocation problem corresponds to a call-scheduling problem where the algorithm must assign starting times to call requests one by one before the first call is established. Hence, the lower bounds in [3] do not apply to the call scheduling problem we study in this paper. Furthermore, their algorithms work for the call-scheduling problem only in the case of unit durations and unit bandwidths.

Scheduling File-Transfers. Coffman *et al.* study a file-transfer scheduling problem that corresponds to call-scheduling in a star with varying edge capacities and calls with unit bandwidth requirements and arbitrary durations [4]. They present complexity results for various restricted versions of the problem, approximation results, and distributed implementations. Many of their results for arbitrary edge capacities and unit bandwidth requirements do not apply to our call-scheduling problem with unit edge capacities and arbitrary bandwidth requirements, however.

Previous Work on On-Line Call-Scheduling. Feldmann *et al.* initiated research on on-line call-scheduling in [7] and [8]. They analyze the GREEDY algorithm (equivalent to LS) and show that running GREEDY once on the calls with bandwidth requirements $\leq \frac{1}{2}$ and once on the calls with bandwidth requirements $> \frac{1}{2}$ yields an on-line algorithm for call-scheduling in binary trees with competitive ratio $12 \log n$. In addition, they obtain results for linear array networks, meshes, complete graphs, and graphs with small separators.

2 Approximation Results for Stars

Stars are the subgraphs of trees that are induced by an arbitrary node of the tree and its neighbors. Hence, call-scheduling problems in stars are encountered as subproblems of call-scheduling in trees. Note that there are two kinds of calls in a star G. First, there are calls that connect the central node to one of the other nodes. Second, there are calls that connect two nodes that are both adjacent to the central node. We refer to these calls as 1-calls and 2-calls, respectively.

In the case of calls with unit durations ($d_r = 1$ for all $r \in R$) and unit bandwidths ($b_r = 1$ for all $r \in R$), call-scheduling in a star is equivalent to edge-coloring a multigraph and thus $\mathcal{NP}$-hard [5]. The algorithm from [13] colors any multigraph G with at most $\lfloor 1.1 \cdot OPT(G) + 0.8 \rfloor$ colors and can be used for the call-scheduling problem with the same performance guarantee, even in trees [5]. The equivalence between call-scheduling and edge-coloring is lost once we allow arbitrary bandwidth requirements or arbitrary call durations. It is known, however, that the performance ratio of LS for call-scheduling with unit bandwidth requirements and arbitrary durations in a star is 2 [4, Corollary 12.2].

2.1 Unit Durations and Arbitrary Bandwidth Requirements

In this section we assume that all call durations are 1, while bandwidth requirements can be arbitrary numbers in $]0;1]$. Note that call-scheduling with arbitrary bandwidth requirements is a generalization of bin-packing and hence $\mathcal{NP}$-complete in the strong sense, even if the network is a single link. Theorem 7, which will be proved in Sect. 2.2, implies that the worst-case performance of LS for calls with unit durations is at most 5. The following tighter result can be proved similar to Lemma 2 below.

Theorem 1. *LS has performance ratio at most* 4.875 *for call-scheduling with arbitrary bandwidth requirements and unit durations in stars.*

Given a schedule S computed by LS for a list L of call requests, it turns out that estimates on the performance ratio of LS on that particular instance L depend heavily on the smallest bandwidth requirement of a call that finishes last in S, i.e., at time $|S|$. The following lemmas make this relationship clearer.

Lemma 2. *Let S be a list-schedule for a list L of calls with arbitrary bandwidth requirements and unit durations. If there is a call r with bandwidth requirement $b_r \leq \frac{1}{2}$ that finishes last in S, then $|S| \leq \lceil 3.875 \cdot OPT(L) \rceil$.*

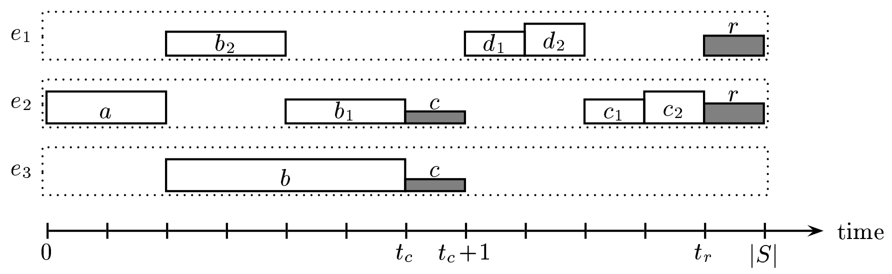

Fig. 1. List-Schedule S, $b_r \leq \frac{1}{2}$, $b_c \leq \frac{1}{3}$

Proof. Assume that r is a 2-call. (If r is a 1-call, the proof is much simpler.) If $b_r \leq \frac{1}{3}$, it follows that at least one of the two edges used by r has less than $\frac{1}{3}$ bandwidth available during at least $\lceil t_r/2 \rceil$ (not necessarily consecutive) time steps prior to t_r. Hence, $OPT > \lceil t_r/2 \rceil \cdot \frac{2}{3} \geq \frac{t_r}{3}$ and $|S| = t_r + 1 \leq 3 \cdot OPT$ in this case. Therefore, assume that $\frac{1}{3} < b_r \leq \frac{1}{2}$. Consider all calls with bandwidth requirement $\leq \frac{1}{3}$ that use at least one edge that is also used by r. Assume that there are such calls, and let c be a call with latest finishing time $t_c + 1$ among them. Furthermore, assume that c is a 2-call and that c uses only one edge that is also used by r. (The cases that no call c exists, that c is a 1-call, or that c is a 2-call using the same edges as r can be treated in a similar way.) Let the edges used by r be e_1 and e_2, and let the edges used by c be e_2 and e_3. Introduce the following variables (cf. Fig. 1): a = (number of) time steps during which c is blocked on e_2; b = time steps during which c is blocked on e_3, but not on e_2; b_1 = time steps prior to t_c during which r is blocked on e_2, but not c; b_2 = time steps prior to t_c during which r is blocked on e_1, but not on e_2; c_1 = time steps after t_c during which r is blocked on e_2 by a single call; c_2 = time steps after t_c during which r is blocked on e_2 by a combination of at least two calls; d_1 = time steps after t_c during which r is blocked on e_1 by a single call, but not blocked on e_2; d_2 = time steps after t_c during which r is blocked on e_1 by a combination of at least two calls, but not blocked on e_2. Note that the time steps accounted for by these variables need not be consecutive. Using these definitions, it is clear that $|S| = a + b_1 + b_2 + c_1 + c_2 + d_1 + d_2 + 2$. If $a + c_2 + d_2 \leq \frac{3}{8} OPT$, the easily observed inequalities $OPT > \frac{2}{3}(b_1 + b_2)$ (follows from $b \geq b_1 + b_2$ and the load on e_3), $OPT \geq c_1 + 1$, and $OPT \geq d_1 + 1$ imply $|S| \leq 3.875 \cdot OPT$. If $a + c_2 + d_2 > \frac{3}{8} OPT$, consider the sum of the loads on e_1 and e_2 (note that there is load $> \frac{1}{2}$ on e_1 or e_2 at time t_c):

$$L(e_1) + L(e_2) > \frac{2}{3}(a + c_2 + d_2) + \frac{1}{2}(b_1 + b_2 + c_1 + d_1 + 1) \tag{1}$$

Since $L(e_1) + L(e_2) \leq 2 \cdot OPT$, we get $|S| - 1 < 4 \cdot OPT - \frac{1}{3}(a + c_2 + d_2) \leq 3.875 \cdot OPT$. Hence, $|S| \leq \lceil 3.875 \cdot OPT \rceil$. □

Lemma 3. *There are stars and lists of calls with arbitrary bandwidth requirements and unit durations such that the schedule computed by LS is longer than the optimum schedule by a factor arbitrarily close to* 3.7. *The call scheduled last by LS has bandwidth requirement* $\frac{1}{2}$.

Proof. We use a well-known worst-case input to first-fit bin-backing (cf. [10, pp. 211-213]) with ratio $\approx \frac{17}{10}$ to construct a call-scheduling input with ratio ≈ 3.7. For any positive integer ℓ divisible by 17, we obtain a list L of calls with optimum schedule length $10\ell/17 + 1$ and list-schedule length $37\ell/17 + 1$. The worst-case input to first-fit bin-packing consists of $30\ell/17$ items with sizes approximately $\frac{1}{6}$, $\frac{1}{3}$, and $\frac{1}{2}$. The optimum packing uses at most $10\ell/17 + 1$ bins for these items, while first-fit requires exactly ℓ bins. Furthermore, each bin is filled to at least $\frac{1}{2} + \delta$ in the packing produced by first-fit, where δ is a parameter that must be chosen sufficiently small.

A list of $30\ell/17$ calls with bandwidth requirements equal to the item sizes in such a bin-packing instance is called a 1.7-*list*. Let $\ell' = 10\ell/17$. The input list L for LS contains calls in a star with $2 + 2\ell' + 3\ell'(\ell' + 1)$ nodes adjacent to the central node c. These nodes are denoted u, $u_1, \ldots, u_{\ell'}$, $v_0, \ldots, v_{\ell'}$, and $w_{i,j}$ for $0 \le i \le \ell'$ and $1 \le j \le 3\ell'$. The list L contains the following calls ($\varepsilon < 1/(6\ell')$):

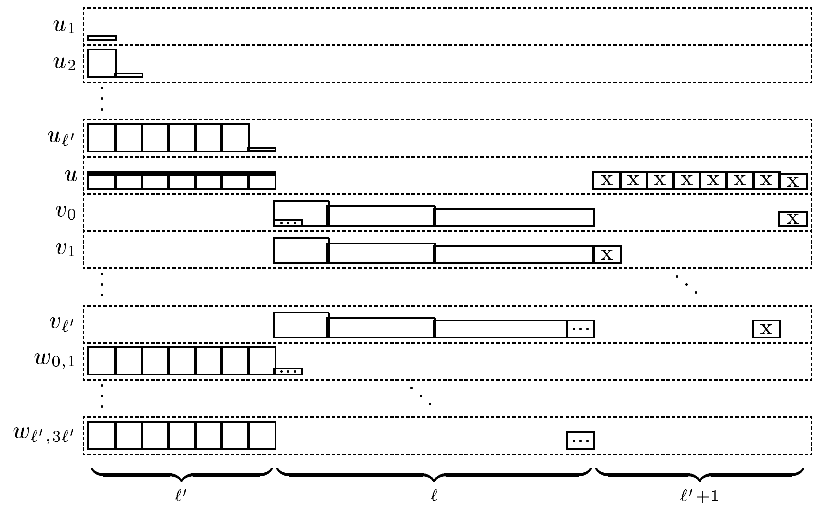

Fig. 2. Example with $LS(L)/OPT(L) \approx 3.7$

1. For $1 \le i \le \ell'$, $i - 1$ calls with bandwidth 1 connecting u_i and c.
2. For $1 \le i \le \ell'$, a call with bandwidth $\frac{1}{2} - \varepsilon$ connecting u and c and a call with bandwidth 3ε connecting u and u_i.

3. For $0 \leq i \leq \ell'$ and $1 \leq j \leq 3\ell'$, ℓ' calls with bandwidth 1 connecting $w_{i,j}$ and c.
4. For $0 \leq i \leq \ell'$, a 1.7-list of calls connecting v_i and some node $w_{i,j}$, such that no two calls connect v_i to the same $w_{i,j}$.
5. For $1 \leq i \leq \ell'$, a call with bandwidth $\frac{1}{2} + \varepsilon$ connecting u and v_i.
6. A call with bandwidth $\frac{1}{2}$ connecting u and v_0.

It is easy to verify that LS will produce the schedule sketched in Fig. 2. The edge $\{u, c\}$ is occupied by a call with bandwidth $\frac{1}{2} - \varepsilon$ and a call with bandwidth 3ε during each of the first ℓ' time steps. All 1.7-lists are scheduled in time steps ℓ' to $\ell' + \ell - 1$, because every call in a 1.7-list is blocked during the first ℓ' time steps on an edge $\{w_{i,j}, c\}$. The calls with bandwidth $\frac{1}{2} + \varepsilon$ connecting u and v_i are scheduled in time steps $\ell' + \ell$ to $2\ell' + \ell - 1$, because they are blocked on $\{u, c\}$ during the first ℓ' time steps and subsequently on $\{v_i, c\}$ during the next ℓ time steps. (Recall that the 1.7-lists occupy at least $\frac{1}{2} + \delta$ bandwidth in time steps ℓ' to $\ell' + \ell - 1$ on all edges $\{v_i, c\}$.) Finally, the call $(u, v_0, \frac{1}{2}, 1)$ is scheduled at time step $2\ell' + \ell$. Hence, $LS(L) = 37\ell/17 + 1$.

On the other hand, it is clear that L can be scheduled in $\ell' + 1$ time steps. In particular, on edge $\{u, c\}$ one can schedule one call with bandwidth $\frac{1}{2} - \varepsilon$ and one call with bandwidth $\frac{1}{2} + \varepsilon$ during each of the first ℓ' time steps. Since ε has been chosen small enough, all calls with bandwidth 3ε together with the call $(u, v_0, \frac{1}{2}, 1)$ can then be scheduled together at time ℓ'. The 1.7-lists can be scheduled in $\ell' + 1$ time steps, such that one of the time steps has bandwidth $\geq \frac{1}{2} + \varepsilon$ available. Hence, the schedule for the 1.7-list on v_i can be arranged such that the call connecting u and v_i is scheduled at that time step. Finally, the remaining 1-calls can be filled in without making the schedule longer. Therefore, $OPT(L) \leq \ell' + 1$. The ratio between $LS(L)$ and $OPT(L)$ is at least $\frac{37\ell+17}{10\ell+17}$, which is arbitrarily close to 3.7 for large ℓ. □

Note that Lemma 2 and Lemma 3 show that the exact bound on the worst-case performance ratio of LS lies between 3.7 and 3.875 if a call with bandwidth requirement $\leq \frac{1}{2}$ finishes last in the list-schedule. Next, we investigate the case that a call with bandwidth requirement $\leq \frac{1}{k}$ for some $k \geq 3$ finishes last in a list-schedule.

Lemma 4. *Let S be a list-schedule for a list L of calls with arbitrary bandwidth requirements and unit durations. If there is a call r with bandwidth requirement $b_r \leq \frac{1}{k}$ for some $k \geq 3$, $k \in \mathbb{N}$, that finishes last in S, then $|S| \leq \left\lceil \frac{2k}{k-1} OPT \right\rceil$.*

Proof. Since r is blocked during the first t_r time steps, at least one of the edges used by r has less than $b_r \leq \frac{1}{k}$ bandwidth available during at least $\lceil t_r/2 \rceil$ time steps. Hence, the load on that edge is greater than $\lceil t_r/2 \rceil \cdot \frac{k-1}{k} < OPT$, and we obtain $t_r < \frac{2k}{k-1} OPT$ and, consequently, $|S| = t_r + 1 \leq \left\lceil \frac{2k}{k-1} OPT \right\rceil$. □

In [10, pp. 217–219], first-fit bin-packing is analyzed under the restriction that all items have size $\leq \alpha$ for some $\alpha \leq \frac{1}{2}$. With $k' = \lfloor 1/\alpha \rfloor$, it is shown that, for any list L of items with sizes $\leq \alpha$, $FF(L) \leq \frac{k'+1}{k'} OPT + 2$ and that there are examples with $FF(L) \geq \frac{k'+1}{k'} OPT - \frac{1}{k'}$. We adapt the construction of these examples to obtain call-scheduling inputs that show that the bound from Lemma 4 is tight.

Lemma 5. *For every $k \geq 3$, $k \in \mathbb{N}$, there are stars and lists L of calls with unit durations and bandwidth requirements $\leq \frac{1}{k-1}$ such that a call with bandwidth requirement $\leq \frac{1}{k}$ finishes last in the list-schedule for L and $\frac{LS(L)}{OPT(L)}$ is arbitrarily close to $\frac{2k}{k-1}$.*

Proof. Let $k' = k-1$. Let ℓ be a positive integer such that k' divides $\ell(k'+1)-1$. We construct a list L of calls with optimum schedule length $\ell + 1$ and list-schedule length $2\frac{\ell(k'+1)-1}{k'}$. Let $b_j^\delta = 1/(k'+1) - k'^{2j+1}\delta$ ($j = 1, 2, \ldots, \ell - 1$) and $a_{1j}^\delta = \cdots = a_{k'j}^\delta = 1/(k'+1) + k'^{2j}\delta$ ($j = 1, 2, \ldots, \ell$), where δ is chosen sufficiently small. A list of calls with exactly one call with bandwidth requirement b_j^δ for each $j = 1, 2, \ldots, \ell - 1$ and one call with bandwidth requirement a_{ij}^δ for each $i = 1, 2, \ldots, k'$ and $j = 1, 2, \ldots, \ell$ is called a *δ-list* if the calls are ordered as follows: the a_{ij}^δ-calls appear in order of non-increasing bandwidths, the b_j^δ-calls appear in order of strictly increasing bandwidths, there are k' a_{ij}^δ-calls between every pair of successive b_j^δ-calls, and the call with bandwidth requirement $b_{\ell-1}^\delta$ is the second call in the list. Note that a δ-list contains $l(k'+1) - 1$ calls.

Consider a δ-list L_δ such that all calls in L_δ are 1-calls using the same edge e. Since first-fit bin-packing is equivalent to LS for calls with unit durations on one edge, [10, pp. 217–219] implies $LS(L_\delta) = \frac{\ell(k'+1)-1}{k'}$ and $OPT(L_\delta) = \ell$. Furthermore, LS schedules exactly k' calls in every time step, and no time step has more than $1/(k'+1) - k'^3\delta$ bandwidth available on edge e in the resulting schedule. In addition, the call with bandwidth $b_1^\delta < \frac{1}{k'+1} = \frac{1}{k}$ is scheduled in the last time step.

We use $\ell(k'+1) - 1$ such δ-lists with 1-calls on separate edges (one edge for each δ-list). These δ-lists come first in the list L. At the end of L, we append one additional δ'-list $L_{\delta'}$, with δ' such that $k'^{2\ell-1}\delta' < k'^3\delta$. Let v be a node of the star that has not been used by any of the 1-calls. The calls in $L_{\delta'}$ all connect the node v to one of the nodes used by the $\ell(k'+1) - 1$ δ-lists, such that no two calls in $L_{\delta'}$ connect v to the same node v'. Obviously, LS will schedule the calls in $L_{\delta'}$ in $\frac{\ell(k'+1)-1}{k'}$ successive time steps starting from $\frac{\ell(k'+1)-1}{k'}$. Hence, the list-schedule has length $2\frac{\ell(k'+1)-1}{k'}$, whereas $OPT = \ell + 1$. Therefore, the performance ratio of LS is arbitrarily close to $\frac{2(k'+1)}{k'} = \frac{2k}{k-1}$. □

While our best general upper bound for the worst-case performance of LS for calls with unit durations and arbitrary bandwidth requirements in stars is 4.875, a slightly modified algorithm gives a much better performance guarantee. The algorithm Decreasing-Bandwidth List-Scheduling (DBLS) behaves just like standard List-Scheduling, but it sorts the given list of call requests according to non-increasing bandwidth requirements before it begins to schedule the calls.

Theorem 6. *DBLS has performance ratio at most $\frac{8}{3}$ for call-scheduling with arbitrary bandwidth requirements and unit durations in stars. There are instances for which the performance ratio of DBLS is arbitrarily close to $\frac{22}{9}$.*

Proof. First, we prove the upper bound. Given a set R of call requests, let $L = L(R)$ be the list of call requests obtained by sorting R in order of non-increasing bandwidth, and denote by S the schedule produced by DBLS. Note that a call c scheduled at time t_c in S is blocked during all time steps prior to t_c entirely by calls that precede c in L. (This holds only because we assume unit call durations.) For a call $c \in R$, denote by L_c the sublist of L that contains all requests from the beginning of the list up to and including c. Taking into account the above argument, it is clear that all calls in L_c are scheduled at the same time step in a list-schedule for L and in a list-schedule for L_c.

We claim that the finishing time $t_c + 1$ of any call $c \in R$ with bandwidth requirement $b_c > \frac{1}{3}$ satisfies $t_c + 1 \le 2 \cdot OPT(R)$. If c has bandwidth requirement $b_c > \frac{1}{2}$, no two calls in L_c can be scheduled at the same time if they use the same edge. Therefore, scheduling L_c is just like scheduling calls with unit bandwidth requirements, and [4, Corollary 12.2] implies $t_c + 1 = LS(L_c) \le 2 \cdot OPT(L_c) \le 2 \cdot OPT(R)$. If c has bandwidth requirement b_c satisfying $\frac{1}{3} < b_c \le \frac{1}{2}$, note that during all time steps prior to t_c more than $1 - b_c$ bandwidth was occupied by other calls from L_c on at least one of the edges used by c. Hence, an edge e was occupied to this extent during at least $\lceil t_c/2 \rceil$ time steps prior to t_c. During each such time step, that edge must have been used either by a single call occupying more than $1 - b_c$ bandwidth or by two calls occupying at least b_c bandwidth each. It is clear that even an optimum schedule requires $\lceil t_c/2 \rceil$ time steps for these calls and an additional time step for c, and thus $t_c + 1 \le 2 \cdot OPT(L_c) \le 2 \cdot OPT(R)$.

Now let r be a call with maximum bandwidth requirement among the calls that finish last in S. If $b_r > \frac{1}{3}$, the previous argument shows that $|S| = t_r + 1 \le 2 \cdot OPT$. If $b_r \le \frac{1}{4}$, note that an edge used by r has less than b_r bandwidth available during at least $\lceil t_r/2 \rceil$ time steps prior to t_r. Hence, $\lceil t_r/2 \rceil \cdot (1 - b_r) < OPT$, implying $t_r < \frac{2}{1-b_r} OPT$ and, therefore, $|S| = t_r + 1 \le \left\lceil \frac{2}{1-b_r} OPT \right\rceil$. With $b_r \le \frac{1}{4}$, this implies $|S| \le \lceil \frac{8}{3} OPT \rceil$.

Finally, consider the case that $\frac{1}{4} < b_r \le \frac{1}{3}$. If r is a 1-call, the edge used by r is occupied to more than $\frac{2}{3}$ during all time steps prior to t_r, and we have $\frac{2}{3} t_r < OPT$, implying $|S| \le \lceil \frac{3}{2} OPT \rceil$. If r is a 2-call, denote by C the set of all calls with bandwidth requirement $> \frac{1}{3}$ that use at least one edge also used by r. If C is empty, r is blocked during the first t_r time steps entirely by calls d with bandwidth requirement b_d satisfying $b_r \le b_d \le \frac{1}{3}$. In addition, it is clear that two such calls are not enough to block r, because $2 \cdot \frac{1}{3} + b_r \le 1$. Therefore, whenever r is blocked on an edge during one of the first t_r time steps, that edge is occupied to at least $3b_r$. Since r is blocked on an edge during at least $\lceil t_r/2 \rceil$ time steps, we have $\lceil t_r/2 \rceil \cdot 3b_r < OPT$. This implies $t_r + 1 \le \left\lceil \frac{2}{3b_r} OPT \right\rceil \le \lceil \frac{8}{3} OPT \rceil$, where the last inequality follows from $b_r > \frac{1}{4}$.

If C is not empty, let c be a call with the latest finishing time among all calls in C. Note that $t_c + 1 \le 2 \cdot OPT$. Furthermore, note that starting from $t_c + 1$

call r is blocked entirely by calls d with bandwidth requirement b_d satisfying $b_r \leq b_d \leq \frac{1}{3}$, and that three such calls are necessary in each time step to block r. Hence, the sum of the loads on the two edges used by r is more than $(t_c+1)(1-b_r)+(t_r-t_c-1)3b_r+2b_r < 2 \cdot OPT$. This can simply be transformed into $3b_r(t_r-t_c-1+\frac{2}{3}+t_c+1) < 2 \cdot OPT+(t_c+1)(4b_r-1)$. Using $t_c+1 \leq 2 \cdot OPT$ and $4b_r-1 \geq 0$, we obtain $t_r+\frac{2}{3} < \frac{8}{3} \cdot OPT$ and, consequently, $t_r+1 \leq \frac{8}{3} \cdot OPT$. This concludes the proof of the upper bound.

Now we give the construction of the instances L that provide the lower bound, using a well-known family of worst-case instances I for first-fit-decreasing bin-packing with $FFD(I) = \frac{11}{9} OPT(I)$ [10, p. 220, Fig. 5.40]. The calls in L have bandwidth requirements $\alpha = \frac{1}{2}+\varepsilon$, $\beta = \frac{1}{4}+2\varepsilon$, $\gamma = \frac{1}{4}+\varepsilon$, and $\delta = \frac{1}{4}-2\varepsilon$. For a given $n \in \mathbb{N}$, let $N = 62208n^5+3888n^3+30n$ and $M = 5184n^4+252n^2+1$, and consider a star with $N+M$ edges $e_1, \ldots, e_N, f_1, \ldots, f_M$. L contains the following calls: (1) for $i = 1, \ldots, N$, we have $6n$ calls with bandwidth α using only edge e_i; (2) for $i = 1, \ldots, M$ and $j = 0, \ldots, 6n-1$, we have one call with bandwidth α using edges f_i and $e_{N-(i-1)\cdot 6n-j}$; (1') for $i = 1, \ldots, N$, we have $6n$ calls with bandwidth β using only edge e_i; (2') for $i = 1, \ldots, M$ and $j = 0, \ldots, 6n-1$, we have one call with bandwidth β using edges f_i and $e_{N-M\cdot 6n-(i-1)\cdot 6n-j}$; (3) for $i = 1, \ldots, N - M \cdot 12n = 864n^3 + 18n$ and $j = 0, \ldots, 6n-1$, we have one call with bandwidth γ using edges e_i and $f_{M-(i-1)\cdot 6n-j}$; (4) for $i = 1, \ldots, M - (864n^3 + 18n) \cdot 6n = 144n^2 + 1$ and $j = 0, \ldots, 6n-1$, we have one call with bandwidth γ using edges f_i and $e_{N-M\cdot 12n-(i-1)\cdot 6n-j}$; (5) for $i = 1, \ldots, 12n$ and $j = 0, \ldots, 12n-1$, we have one call with bandwidth δ using edges e_i and $f_{2+(i-1)12n+j}$; (6) for $j = 0, \ldots, 12n-1$, we have one call with bandwidth δ using edges f_1 and e_{1+j}. It is not difficult to show that $DBLS(L) = 22n$ and $OPT(L) = 9n + 1$. An optimum schedule can combine calls such that the full capacity of the edges is exploited most of the time. (Note that $\alpha+\gamma+\delta = 1$ and $2\beta + 2\delta = 1$.) Details are omitted. $\square$

2.2 Arbitrary Durations and Arbitrary Bandwidth Requirements

In this section, call durations can be arbitrary positive integers, and bandwidth requirements can be arbitrary numbers in $]0;1]$.

Theorem 7. *If S is the schedule computed by LS for a list L of call requests with arbitrary durations and bandwidth requirements in a star, then $LS(L) \leq 5 \cdot OPT(L)$. If there is a call with bandwidth requirement $\leq \frac{1}{2}$ that finishes last in S, then $|S| \leq 4 \cdot OPT(L)$. If there is a call with bandwidth requirement $\leq \frac{1}{3}$ that finishes last in S, then $|S| \leq 3 \cdot OPT(L)$.*

Proof. Let r be a call with the smallest bandwidth requirement b_r among all calls that finish last in S, i.e., at time $|S|$. Since call r is blocked during all time steps prior to t_r, the load on at least one edge used by r is more than $\lceil t_r/2 \rceil \cdot (1-b_r) + d_r b_r < OPT$. This implies $t_r \leq \frac{2}{1-b_r} OPT - \frac{2b_r}{1-b_r} d_r$, and we obtain $t_r + d_r \leq \frac{2}{1-b_r} OPT + \frac{1-3b_r}{1-b_r} d_r$. For $b_r \leq \frac{1}{3}$, we have $1-3b_r \geq 0$ and, using

$d_r \leq OPT$, obtain $|S| = t_r + d_r \leq \frac{2+(1-3b_r)}{1-b_r} OPT = 3 \cdot OPT$; for $\frac{1}{3} < b_r \leq \frac{1}{2}$, $\frac{2}{1-b_r}$ is at most 4, and with $1 - 3b_r < 0$ we obtain $|S| = t_r + d_r \leq 4 \cdot OPT$.

Assume now that $b_r > \frac{1}{2}$ and that r is a 2-call; if r is a 1-call, similar arguments can be applied. Consider all calls with bandwidth requirement $\leq \frac{1}{2}$

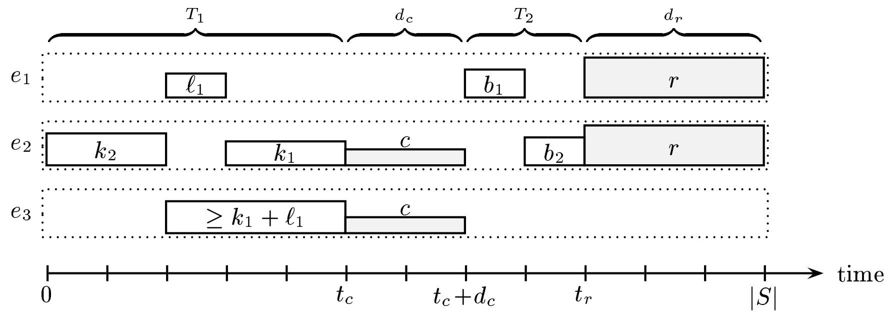

Fig. 3. List-Schedule S, $b_r > \frac{1}{2}$, $b_c \leq \frac{1}{2}$

that use at least one edge that is also used by r. If there is no such call, at least one of the edges used by r is blocked by a call with bandwidth $> \frac{1}{2}$ in at least $\lceil t_r/2 \rceil + d_r \leq OPT$ time steps and, consequently, $|S| = t_r + d_r \leq 2 \cdot OPT$. Otherwise, let c be a call with the latest finishing time $t_c + d_c$ among all such calls. Assume that c is a 2-call that uses only one edge that is also used by r. (The cases that c is a 1-call and that c is a 2-call that uses the same edges as r can be treated similarly.) Furthermore, assume that c finishes before r is established. (Otherwise, $|S| \leq 5 \cdot OPT$ follows directly from $t_c + d_c \leq 4 \cdot OPT$ and $d_r \leq OPT$.) Let the edges used by r be e_1 and e_2, and let the edges used by c be e_2 and e_3. The list-schedule is partitioned into the following disjoint time intervals: (A) T_1 time steps from the beginning of the schedule until t_c (the time when call c is scheduled), (B) d_c time steps during which call c is active, (C) T_2 time steps from the finishing time of c until t_r (the time when call r is scheduled), and (D) d_r time steps during which call r is active. Obviously, $|S| = T_1 + d_c + T_2 + d_r$. Introduce the following variables (cf. Fig. 3): b_1 = number of time steps in part (C) during which r is blocked on e_1, but not on e_2; b_2 = number of time steps in part (C) during which r is blocked on e_2; k_1 = number of time steps in part (A) during which r is blocked on e_2, but not c; k_2 = number of time steps in part (A) during which c (and r) is blocked on e_2; ℓ_1 = number of time steps in part (A) during which r is blocked on e_1, but not on e_2. Considering the load on edge e_3, we obtain $(k_1 + \ell_1)(1 - b_c) + d_c b_c \leq OPT$. This implies $k_1 + \ell_1 \leq \frac{1}{1-b_c} OPT - \frac{b_c}{1-b_c} d_c$. Adding d_c on both sides of this inequality, we get $k_1 + \ell_1 + d_c \leq \frac{1}{1-b_c} OPT + \frac{1-2b_c}{1-b_c} d_c$. Since $d_c \leq OPT$ and $1 - 2b_c \geq 0$, this implies $k_1 + \ell_1 + d_c \leq \frac{1+(1-2b_c)}{1-b_c} OPT = 2 \cdot OPT$. In addition, it is easy to observe that $k_2 + b_2 \leq 2 \cdot OPT$ and $b_1 + d_r \leq OPT$. Taking into account that $|S| = k_1 + k_2 + \ell_1 + d_c + b_1 + b_2 + d_r$, these inequalities can be combined to obtain $|S| \leq 5 \cdot OPT$. □

For the case that a call with bandwidth requirement $\leq \frac{1}{3}$ finishes last in a list-schedule, the following lemma shows that the upper bound 3 on the worst-case performance of LS is tight.

Lemma 8. *For arbitrary $k > 2$, $k \in \mathbb{N}$, there are stars and lists of call requests with arbitrary durations and bandwidth requirements $\leq \frac{1}{k-1}$ such that a call with bandwidth requirement $\leq \frac{1}{k}$ finishes last in the list-schedule and the performance ratio of LS is arbitrarily close to 3.*

Proof. Fix arbitrary integers $k > 2$ and $\ell > 1$. We construct a list L of call requests such that $LS(L) = 3\ell$, $OPT(L) = \ell + 1$, and the call that finishes last in the list-schedule for L has bandwidth requirement $\frac{1}{k}$.

The star used for the construction has $k\ell + 3$ nodes: the central node c and nodes u, v, $u_1, \ldots, u_\ell$, $v_1, \ldots, v_{(k-1)\ell}$ adjacent to c. The list L contains the following call requests ($\varepsilon \ll 1$):

(1) For $i = 1, \ldots, \ell$: $k-1$ calls $(u, c, \frac{1}{k}, 1)$, $k(i-1)$ calls $(u_i, c, \frac{1}{k}, 1)$, and one call $(u_i, u, \varepsilon, 1)$.
(2) For $i = 1, \ldots, (k-1)\ell$: $k\ell$ calls $(v_i, c, \frac{1}{k}, 1)$.
(3) For $i = 0, \ldots, \ell-1$: for $j = 1, \ldots, k-1$: one call $(v, v_{i(k-1)+j}, \beta_{i,j}, 1)$.
(4) One call $z = (u, v, \frac{1}{k}, \ell)$.

The bandwidth requirements $\beta_{i,j}$ are defined by $\beta_{i,1} = \frac{1}{k} + k\delta_i$ and $\beta_{i,2} = \cdots = \beta_{i,k-1} = \frac{1}{k} - \delta_i$, where $\delta = \delta_0$ is chosen sufficiently small and $\delta_{i+1} = \frac{k-2}{k}\delta_i$.

What schedule is produced by LS for the list L? The calls (1) fill the edge $\{u, c\}$ to $\frac{k-1}{k} + \varepsilon$ during the first ℓ time steps. Each of the calls with bandwidth ε is blocked on one of the edges $\{u_i, c\}$ in all time steps before its starting time. The calls (2) fill the edges $\{v_i, c\}$ completely during the first ℓ time steps. The calls (3) are scheduled in time steps ℓ to $2\ell - 1$, because each call is blocked on a different edge $\{v_i, c\}$ during the first ℓ time steps and blocked on the edge $\{v, c\}$ from time step ℓ up to its starting time. Exactly $k-1$ calls (3) are scheduled in each time step, because their bandwidths add up to $\frac{k-1}{k} + 2\delta_i$ and, therefore, block all subsequent calls (3). Finally, call z is scheduled at time 2ℓ, because it is blocked on $\{u, c\}$ during the first ℓ time steps and on $\{v, c\}$ during the second ℓ time steps. Hence, $LS(L) = 3\ell$.

In an optimum schedule, call z is scheduled at time 0. In each of the first ℓ time steps, $k-1$ calls from (1) using edge $\{u, c\}$ and with bandwidth requirement $\frac{1}{k}$ can be scheduled together with z. All the calls from (1) with bandwidth ε are scheduled together at time ℓ. The remaining calls from (1) can easily be scheduled in free time slots during the first ℓ time steps.

Among the calls from (3), the $k-1$ calls with bandwidth requirements $\beta_{i,2}, \ldots, \beta_{i,k-1}, \beta_{i+1,1}$ are scheduled together in time step i, for $0 \leq i \leq \ell - 1$. (For $i = \ell - 1$, there is no call with bandwidth $\beta_{i+1,1}$, and only $k-2$ of the calls from (3) are scheduled in time step $\ell - 1$.) The bandwidths of the calls from (3) scheduled during one of the time steps $0, \dots, \ell-1$ add up to at most $\frac{k-1}{k}$. Hence, they can be scheduled concurrently with call z. The call with bandwidth $\beta_{0,1}$ is scheduled at time ℓ. The calls from (2) can easily be scheduled in the remaining free time slots during the first $\ell + 1$ time steps. Therefore, $OPT = \ell + 1$. $\square$

3 Approximation Results for Trees

3.1 Unit Durations and Arbitrary Bandwidth Requirements

It is known that the performance of LS can be arbitrarily bad in trees or even in chains if arbitrary bandwidth requirements are allowed. Feldmann *et al.* give a list of call requests with unit durations on a chain with $n+1$ nodes such that the performance ratio of LS is $\Omega(n)$ [7]. Therefore, we consider a variation of the basic List-Scheduling algorithm. Pick an arbitrary node of the tree network as the root and assign each node of the tree a *level* according to its distance from the root. (The root has level 0.) Let m_r be that node on P_r (the path corresponding to call r) whose level is minimum among all nodes on P_r. The level of a call r is defined to be equal to the level of the node m_r. We consider the Level-List-Scheduling algorithm (LLS), which is identical to List-Scheduling except that it sorts the list of calls according to non-decreasing levels before it starts to schedule the calls.

Theorem 9. *LLS has performance ratio at most* 6 *for call-scheduling with arbitrary bandwidth requirements and unit durations in trees.*

Proof. Let S be a schedule computed by LLS for a given set R of call requests. First, we show that any call r with bandwidth requirement $b_r \leq \frac{1}{2}$ finishes no later than at time $4 \cdot OPT$. To see this, consider the node m_r, and let e_1 and e_2 be the edges incident to m_r that are used by r. (If r uses only one edge incident to m_r, it can be proved by similar arguments that $t_r + 1 \leq 2 \cdot OPT$.) It is clear that call r is blocked either on edge e_1 or on edge e_2 by calls with equal or smaller level during all time steps prior to t_r. Hence, at least one of these edges has less than $\frac{1}{2}$ bandwidth available during at least $\lceil \frac{t_r}{2} \rceil$ time steps prior to t_r. Therefore, $OPT > \frac{1}{2} \cdot \lceil \frac{t_r}{2} \rceil$ and, consequently, $t_r + 1 \leq 4 \cdot OPT$.

Now, let r be a call with minimum bandwidth requirement b_r among all calls that finish last in S. If $b_r \leq \frac{1}{2}$, the argument above implies $|S| \leq 4 \cdot OPT$. Therefore, assume that $b_r > \frac{1}{2}$. Let e_1 and e_2 be the edges incident to m_r that are used by r. Again, it is clear that call r is blocked either on edge e_1 or on edge e_2 by calls with equal or smaller level during all time steps prior to t_r. Let c be a call with bandwidth requirement $b_c \leq \frac{1}{2}$ that has the latest finishing time among all such calls. (If no such call exists, call r is blocked only by calls with smaller or equal level and with bandwidth requirements $> \frac{1}{2}$, and $|S| \leq 2 \cdot OPT$.) The argument above implies $t_c + 1 \leq 4 \cdot OPT$, and $t_r - t_c \leq 2 \cdot OPT$ follows from the fact that call r is blocked by calls with bandwidth requirements $> \frac{1}{2}$ either on e_1 or on e_2 during all time steps from $t_c + 1$ to t_r. Combining these inequalities, we obtain $|S| = t_r + 1 \leq 6 \cdot OPT$. $\square$

3.2 Arbitrary Durations and Arbitrary Bandwidth Requirements

Given a tree network T with n nodes, we use a well-known technique [2] based on a tree separator [16] to assign levels to the nodes of T as follows:

1. Choose a node v whose removal splits T into subtrees $T_1, T_2, \dots, T_k$ with at most $n/2$ nodes each. Assign node v the level 0.
2. In each subtree T_i with n_i nodes, find a node v_i whose removal splits T_i into subtrees with at most $n_i/2$ nodes. Assign all such nodes v_i the level 1.
3. Continue recursively until every node of T is assigned a level.

This way every node of T is assigned a level ℓ, $0 \leq \ell \leq \log n$. For each call request $r = (u, v, b, d)$ in T, the level of r is defined to be the smallest level of all nodes on the path P_r from u to v. In addition, the *root* node of r is defined to be that node on P_r whose level is equal to the level of r. (Note that the root node is uniquely determined; if two nodes of equal level are on a path P, there must exist a node of smaller level on P.) Given a list L of call requests in T, let L_ℓ be the sublist of L that contains all call requests of level ℓ, $0 \leq \ell \leq \log n$. Note that scheduling a list L_ℓ is equivalent to scheduling calls in a number of disjoint stars: calls in L_ℓ with the same root node intersect if and only if they use the same edge incident to that root node; calls in L_ℓ with different root nodes never intersect. Therefore, $LS(L_\ell) \leq 5 \cdot OPT(L_\ell)$ as a consequence of Theorem 7. The algorithm List-Scheduling by Levels (LSL) simply uses List-Scheduling to schedule the lists L_ℓ, $0 \leq \ell < \log n$ one after another. ($L_{\log n}$ is empty, because the root node of a call can never have level $\log n$.) LSL begins to schedule $L_{\ell+1}$ only when all calls from L_ℓ have finished. Note that LSL is an on-line algorithm because it does not require advance knowledge of call durations. Hence, we obtain the following theorem:

Theorem 10. *LSL is an on-line algorithm for scheduling calls with arbitrary bandwidth requirements and arbitrary durations in trees. Its competitive ratio is at most* $5 \log n$.

4 Conclusion

We have analyzed List-Scheduling and variants of it for the call-scheduling problem in stars and trees. It was shown that variants of LS have good, constant performance ratio in all cases except for call-scheduling with arbitrary bandwidths and arbitrary durations in trees, where the ratio is $5 \log n$. Hence, List-Scheduling variants, which are easy to implement, can be applied in practice to schedule connections in networks with guaranteed quality of service.

Regarding possible directions for future research, it will be interesting to study call-scheduling algorithms for the cases that edge capacities may vary, that directed and undirected calls as well as calls with release times are allowed, and that the topology of the network is such that multiple paths between the endpoints of each connection exist.

References

1. Y. Aumann and Y. Rabani. Improved bounds for all optical routing. In *Proceedings of the 6th Annual ACM-SIAM Symposium on Discrete Algorithms SODA '95*, pages 567–576, 1995.
2. B. Awerbuch, Y. Bartal, A. Fiat, and A. Rosén. Competitive non-preemptive call control. In *Proceedings of the 5th Annual ACM–SIAM Symposium on Discrete Algorithms SODA '94*, pages 312–320, 1994.
3. Y. Bartal and S. Leonardi. On-line routing in all-optical networks. In *Proceedings of the 24th International Colloquium on Automata, Languages and Programming ICALP '97*, LNCS 1256, pages 516–526. Springer-Verlag, 1997.
4. E. Coffman, Jr., M. Garey, D. Johnson, and A. Lapaugh. Scheduling file transfers. *SIAM J. Comput.*, 14(3):744–780, August 1985.
5. T. Erlebach and K. Jansen. Scheduling of virtual connections in fast networks. In *Proceedings of the 4th Parallel Systems and Algorithms Workshop PASA '96*, pages 13–32. World Scientific Publishing, 1997.
6. T. Erlebach and K. Jansen. Call scheduling in trees, rings and meshes. In *Proceedings of the 30th Hawaii International Conference on System Sciences HICSS-30*, volume 1, pages 221–222. IEEE Computer Society Press, 1997.
7. A. Feldmann, B. Maggs, J. Sgall, D. 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.
8. 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.
9. R. Graham. Bounds on multiprocessing timing anomalies. *SIAM J. Appl. Math.*, 17(2):416–429, March 1969.
10. R. Graham. Bounds on the performance of scheduling algorithms. In E. G. Coffman, Jr., editor, *Computer and Job-Shop Scheduling Theory*, pages 165–227. John Wiley & Sons, Inc., New York, 1976.
11. J. Hoogeveen, S. van de Velde, and B. Veltman. Complexity of scheduling multiprocessor tasks with prespecified processor allocations. *Discrete Appl. Math.*, 55:259–272, 1994.
12. C. Kaklamanis, P. Persiano, T. Erlebach, and K. Jansen. Constrained bipartite edge coloring with applications to wavelength routing. In *Proceedings of the 24th International Colloquium on Automata, Languages and Programming ICALP '97*, LNCS 1256, pages 493–504. Springer-Verlag, 1997.
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. D. B. Shmoys, J. Wein, and D. P. Williamson. Scheduling parallel machines online. In *Proceedings of the 32nd Annual Symposium on Foundations of Computer Science FOCS '91*, pages 131–140, 1991.
15. The ATM Forum, Upper Saddle River, NJ. *ATM User-Network Interface (UNI) Specification Version 3.1.*, 1995.
16. J. van Leeuwen, editor. *Handbook of Theoretical Computer Science. Volume A: Algorithms and complexity.* Elsevier North-Holland, Amsterdam, 1990.