A Preemptive Algorithm for Maximizing Disjoint Paths on Trees

Yossi Azar * Uriel Feige † Daniel Glasner ‡

October 15, 2008

Abstract

We consider the online version of the maximum vertex disjoint path problem when the underlying network is a tree. In this problem, a sequence of requests arrives in an online fashion, where every request is a path in the tree. The online algorithm may accept a request only if it does not share a vertex with a previously accepted request. The goal is to maximize the number of accepted requests. It is known that no online algorithm can have a competitive ratio better than $\Omega(\log n)$ for this problem, even if the algorithm is randomized and the tree is simply a line. Obviously, it is desirable to beat the logarithmic lower bound. Adler and Azar [SODA 1999] showed that if preemption is allowed (namely, previously accepted requests may be discarded, but once a request is discarded it can no longer be accepted), then there is a randomized online algorithm that achieves constant competitive ratio on the line. In the current work we present a randomized online algorithm with preemption that has constant competitive ratio on any tree. Our results carry over to the related problem of maximizing the number of accepted paths subject to a capacity constraint on vertices (in the disjoint path problem this capacity is 1). Moreover, if the available capacity is at least 4, randomization is not needed and our online algorithm becomes deterministic.

1 Introduction

We consider the online version of the maximum vertex disjoint paths problem, and of paths selection subject to congestion (a.k.a. capacity) constraints. Given a communication network which is a connected graph $G = (V, E)$ (where $|V| = n$), the on-line algorithm processes a sequence of call requests. Each request specifies a pair of vertices $(v, w) \in V \times V$. When a request arrives the algorithm can accept it by allocating a path connecting v and w in G, or reject it. The goal is to maximize the number of accepted calls in such a way that the allocated paths conform with the congestion constraints.

The performance of an on-line algorithm is measured by its competitive ratio. A deterministic or randomized on-line algorithm is ρ-competitive if for any input sequence its (expected) benefit is not less than $1/\rho$ times the benefit of an optimal off-line solution.

*Tel Aviv University, Tel Aviv, Israel, and Microsoft Research, Redmond Washington. azar@tau.ac.il. Research supported in part by the Israel Science Foundation.

†Weizmann Institute, Rehovot, Israel. uriel.feige@weizmann.ac.il

‡Weizmann Institute, Rehovot, Israel. dglasner@gmail.com

A preemptive algorithm is an algorithm which is allowed to preempt accepted calls. Such an algorithm may decide at any point to discard any number of calls which it has already accepted. These calls may not be recalled at a later time and do not count towards the algorithm's benefit.

Versions of this problem and its generalization, the *call control* problem, in which call requests also have varying *bandwidth* and *benefit* specifications, have been extensively studied. See for example [4, 6, 8], for surveys of the problem see [10] and [15].

Our results: We present a randomized preemptive algorithm for the on-line maximum vertex disjoint paths problem on trees, and show that it has constant competitive ratio. Our result is best possible in the sense that if one disallows either randomization or preemption, then every online algorithm cannot be better than $\Omega(\log n)$ competitive, even on line networks [12, 7, 17]. We also extend our result to maximizing the number of paths subject to a congestion bound of b for all $b > 1$. When $b \geq 4$, our algorithm can be made deterministic. For any b, preemption is still provably required if one is to achieve a constant competitive ratio.

Previously, $\Theta(\log D)$ competitive algorithms were known for trees where D is the diameter of the tree (see [8] and [16]). Those algorithms are non-preemptive. A constant competitive algorithm was known only for the line network [1], and as noted above, it is unavoidable that the online algorithm achieving this is preemptive and randomized. That algorithm can be made deterministic when a congestion bound of $b \geq 2$ is given.

Related work: There are numerous versions of the disjoint path problem, depending on whether graphs are directed or undirected (we consider undirected graphs), capacity constraints are on edges or vertices (we assume that they are on vertices), requests arrive as paths or as source-destination pairs and the algorithm may choose the path (for trees this does not matter, and for general graphs we assume that requests are source-destination pairs), algorithms are online or off-line (we consider the online case), algorithms are randomized or deterministic (we allow randomization), whether preemption is allowed in online settings (we allow preemption), and whether the underlying graph can be arbitrary or has some special structure (we consider trees). For lack of space, we shall mention only those results that we find most informative to our current setting.

Off-line setting: For small capacity bound b on edges, there is no polynomial time constant approximation [3] (unless NP has quasi-polynomial time algorithms) for a general network. In contrast, if the capacity bound is more than logarithmic then randomized rounding of a linear programming relaxation gives a $(1 + \epsilon)$ approximation for maximizing the number of paths [18], and this holds regardless of whether capacity constraints are placed on edges or vertices.

On trees the maximum edge disjoint paths is solvable in polynomial time, and becomes NP-hard when edges have a capacity bound $b \geq 2$ [13]. We observe here that the maximum paths problem in trees with vertex capacity bound b is solvable using dynamic programming in time $n^{b+O(1)}$, and becomes NP-hard only when b grows as a function of n.

Online setting: When the capacity bound is $b \geq \log n$, the deterministic non-preemptive algorithm of [6] is $O(\log n)$ competitive on a general network. This is the best possible even

among randomized algorithms, in the sense that there is a lower bound of $\Omega(\log n)$ for non-preemptive algorithms for any allowed congestion even for a line network. For the disjoint paths problem (i.e. congestion is $b = 1$) on a general network, an $\Omega(n^{\epsilon})$ lower bound was shown in [9] even for randomized preemptive algorithms. This lower bound is not known to extend to the case where $b \geq 2$. When the requests are paths rather than source-destination pairs and the capacity constraint is $b - 1$, there is an $\Omega(n^{1/b}/b)$ lower bound on the competitive ratio of deterministic preemptive and randomized non-preemptive algorithms, and an $\Omega(n^{1/(2b)}/b)$ lower bound for randomized preemptive algorithms [2]. (Lower bounds for the case when requests are paths involve requests that need not resemble shortest paths.)

For some specific networks such as trees, meshes and classes of planar graphs (see [7, 8, 14]) there are known non-preemptive algorithms with $O(\log n)$ competitive ratios for the disjoint paths problem.

It still remains open whether a sub-logarithmic (randomized or deterministic) preemptive algorithm exists for general networks when we allow high congestion. Our result shows that this is possible for trees even when the congestion is low.

Overview of the paper: In section 2 we introduce some definitions and notation. In section 3, which is the main section, we present a deterministic preemptive algorithm with constant competitive ratio. However, this algorithm assumes that the vertices have a capacity of 4 rather than 1. In section 4 we use randomization in order to remove the assumption on capacity, and thus derive a randomized preemptive algorithm for the disjoint paths problem. The extension of our results to the capacity b case is discussed in section 4.1.

Our techniques: The approach followed in Section 3 is to decompose the tree problem to a sum of independent subproblems on line networks, and then on each subproblem to use the algorithm from [1] that has a constant competitive ratio on the line. Namely, in an online fashion our algorithm attempts to partition the requests into subsequences. With each subsequence it associates one path (hence, a line network) in the tree, and the subsequences have the property that all requests for the same subsequence intersect the path that is associated with the subsequence, and do not intersect any request from any other subsequence. Achieving a partition with the above property is in general impossible, so our online algorithm will need to drop some of the requests, so as to be able to partition the remaining sequence of requests into subsequences. Our analysis will show that the approximation ratio does not suffer much because of these dropped requests. An additional source of difficulty is that the line algorithm from [1] cannot be applied as is to a subsequence. The reason for this is that the partition to subsequences is dynamic and is not known in advance, and hence the path associated with a subsequence is also not fixed in advance. We overcome this problem by partitioning each subsequence into two groups. In one group, corresponding to the part of the path that is already fixed, we apply the line algorithm of [1]. In the other group, corresponding to the part of the path that may still grow dynamically, we apply a new on-line algorithm which follows the behavior of the off-line algorithm for the *activity selection* problem.

2 Preliminaries

We consider a network T which is a tree. By choosing an arbitrary vertex r we root the tree. A call request is characterized by two distinct nodes, since the underlying network is a tree a call request defines a single path. We denote the input sequence of call requests by σ and will refer to the call requests as paths.

Without loss of generality we can assume that all call requests are from a leaf to a leaf. This can be achieved by adding a new node for each internal node of T and connecting it to its corresponding node.

The length of the path from the root r to a node v is the *depth* of v in T. We define the *least depth node* of a path P on a rooted tree, denoted by $ldn(P)$, as the node with the least depth in P. A *monotonic path* P on a rooted tree, is a path with a sequence of node depths which is monotonic. Any path P which is not monotonic is comprised of two monotonic paths which intersect at $ldn(P)$. Having fixed some arbitrary orientation, we call them $left(P)$ and $right(P)$. We define the *maximal depth node* of a monotonic path P on a rooted tree, denoted by $mdn(P)$, as the node with the maximal depth in P. When the context is clear we will sometimes use the notation $v < w$ for two nodes v and w meaning $depth(v) < depth(w)$. The notation $[v, w)$ will be used for the path connecting nodes v and w, excluding w.

Let σ be a sequence of requested calls. The *congestion* created by a set of calls $C \subseteq \sigma$ on a node v is the number of paths in C which intersect v. The congestion created by a set of calls $C \subseteq \sigma$ on a subgraph $H \subseteq T$ is the maximal congestion created by C on the nodes of H. The maximal congestion created by an on-line algorithm $\mathcal{A}$ is the maximal congestion created by $\mathcal{A}(\sigma)$ on T for all input sequences σ, where $\mathcal{A}(\sigma) \subseteq \sigma$ are the calls accepted by $\mathcal{A}$. Given a bound b on the maximal congestion, we say that an algorithm or a set of calls is *b-congested* if the maximal congestion created by it on T is bounded by b.

The performance of a b-congested randomized on-line algorithm $\mathcal{A}$, is measured in terms of its competitive ratio, defined as follows. Let $\mathcal{OPT}_\sigma \subseteq \sigma$ be a maximal size b-congested subset. We say that randomized A is *ρ-competitive* if for all request sequences σ we have $E(|\mathcal{A}(\sigma)|) \geq \frac{1}{\rho}|\mathcal{OPT}_\sigma|$. During the analysis we will compare the performance of deterministic b-congested on-line algorithms on an input sequence σ to a maximal size 1-congested subset (which is in fact a subset of disjoint calls). We denote such a selection by $\mathcal{OPT}_\sigma^{(1)} \subseteq \sigma$ and say that A is *ρ-competitive against a 1-congested optimal selection* if for all request sequences σ we have $|\mathcal{A}(\sigma)| \geq \frac{1}{\rho}|\mathcal{OPT}_\sigma^{(1)}|$.

While describing the on-line algorithm we will use the following terms. A call which is discarded at its arrival time is a *rejected* call, a call which is discarded after it has been accepted is a *preempted* call. Calls which are either rejected or preempted are *discarded* calls.

Some of the objects we will discuss evolve as a function of the input requests. We will use the notation O^* for such an object O, to denote its final state.

3 A 4-congested deterministic algorithm

In this section we present a deterministic on-line algorithm whose maximal congestion does not exceed 4. We will also show that it is 6 competitive against a 1-congested optimal

solution on the same request sequence.

Overview: The algorithm dynamically partitions the incoming calls into subsequences σ^i for $i = 1, \ldots, k$. The number of subsequences k, is not known in advance and increases over time. This partitioning is described in subsection 3.1. An algorithm for processing the calls in a single subsequence is given in subsection 3.2. The algorithm for combining the selections made on each subsequence into a global selection is discussed in subsection 3.3.

3.1 Partitioning σ into subsequences and maintaining the stem structure

Definition 3.1 *Let $S \subseteq T$ be the subtree connecting the least depth nodes of the calls in σ and r, where r is the root of T. A* stem structure for σ *is a partition of S into node disjoint monotonic paths such that the maximal depth node in each path is a leaf of S. Each path (with one exception) is half open, i.e., it contains its maximal depth node but does not contain its least depth node. One path that contains the root r is closed, i.e., it contains both its maximal depth node and least depth node.*

Given a stem structure for σ we denote the closed path that contains r, by $stem^1$. We number the half open paths $2, \ldots, k$ and refer to the i'th such path as $stem^i$. The node incident in $stem^i$'s open edge which does not belong to $stem^i$ is called the *root* node of $stem^i$.

The stem structure has a tree hierarchy. Specifically, $stem^1$ is the root stem and for all other stems, a stem's parent is the stem that contains its root node.

Using a stem structure we can partition the calls in σ into subsequences. The calls whose least depth node lies in $stem^i$ are the calls in σ^i. Note that $stem^i$ is a monotonic path that connects the least depth nodes of the calls in σ^i, thus providing a line network structure.

We will use the procedure *StemStructure* described in figure 1 to create and maintain a stem structure for σ and partition the calls accordingly in an online fashion.

Claim 3.2 *Algorithm StemStructure maintains a stem structure for σ.*

Proof: We prove by induction on the calls in σ. The base case is trivial since the stem structure of an empty subsequence is r. For the inductive step we consider the arrival of a new call P. By the inductive assumption the algorithm has maintained a stem structure for the calls that have arrived so far. Let S be the tree connecting the least depth nodes of these calls and r, and let S' be the tree which also connects $ldn(P)$. If $ldn(P)$ lies on S, the stem structure does not change. Otherwise let $[ldn(P), v)$ be the path that connects $ldn(P)$ to S. If $v = mdn(stem^j)$ for some j then $stem^j \cup [ldn(P), v)$ is a monotonic path and we will extend $stem^j$ so that its maximal depth node will now be $ldn(P)$ which is a leaf of S'. If v is some inner node of S we will go to the else clause in the procedure and create a new stem $[ldn(P), v)$. This new stem covers $S' \setminus S$, it is a monotonic path and its least depth node is $ldn(P)$ which is a leaf of S' $\square$

In fact the procedure generates a sequence of stem structures as a function of σ. We note that when further calls arrive the stem structure never "shrinks". In particular, for all i and for each arriving call, $stem^i$ before the arrival of the call is contained in $stem^i$ as

Procedure: *StemStructure*
Initialize: $i \leftarrow 1$, $\sigma^1 \leftarrow \emptyset$, $stem^1 \leftarrow r$
for each incoming call $P \in \sigma$
Starting at $ldn(P)$ traverse the path to r until reaching a node v belonging to $stem^j$ for some j
if $stem^j \cup [ldn(P), v)$ is a monotonic path
$\sigma^j \leftarrow \sigma^j \cup P$
$stem^j \leftarrow stem^j \cup [ldn(P), v)$
else
$\sigma^{i+1} \leftarrow P$
$stem^{i+1} \leftarrow [ldn(P), v)$
$i \leftarrow i + 1$
end if
end for

Figure 1: Algorithm for partitioning the calls and maintaining the stems

modified by the call. This implies that $depth(mdn(stem^i))$ is a non decreasing sequence. Furthermore, existing stems are never removed, only new stems may be added. A stem's root node is fixed and does not change once the stem has been created. The stem's parent and ancestor stems are fixed at the moment of its creation but descendent stems may be created later on.

3.2 An algorithm for subsequence σ^i

In this subsection we consider the processing of the calls in a single subsequence σ^i competing against a 1-congested optimal selection on these calls only.

In an off-line setting, by considering the intersection of the calls in σ^{*i} (the final state of σ^i) with the appropriate stem, $stem^{*i}$ (the final state of $stem^i$) we can reduce the problem to a line network.

Lemma 3.3 *Let $C \subseteq \sigma^{*i}$, if all calls in C have a common (non-empty) intersection and $ldn(P)$ is of maximal depth in $\{ldn(Q)|Q \in C\}$ then $ldn(P) \in \bigcap_{Q \in C} Q$.*

Proof: Whenever two paths P and Q intersect $ldn(P \cap Q) = \max(ldn(P), ldn(Q))$. Since $ldn(P)$ is maximal in $\{ldn(Q)|Q \in C\}$, all calls in C intersect $ldn(P)$. □

Corollary 3.4 *A bound on the maximal congestion created by σ^i on $stem^i$ is also a bound on the congestion created by the calls in σ^i anywhere on T.*

We assume that we are given an algorithm *Line* for maximizing vertex disjoint paths on a line network. Specifically, a 2-congested algorithm for maximizing edge disjoint paths on a line was shown in [1]. It is 2 competitive against a 1-congested optimal selection.

The natural approach would be to reduce the tree problem to several line problems and apply the line algorithm on each one separately using the corollary above. A difficulty

which arises in the on-line setting is that $stem^{*i}$ is not known in advance. Specifically, even after a call has been assigned to a subsequence its intersection with $stem^{*i}$ is not always known. This uncertainty rules out a straightforward reduction to an on-line algorithm for a line network.

For example, the known algorithm for the line has the property that it preempts a containing call in favor of the contained call. However, in reducing the tree to lines the containment relationship may become uncertain when the calls intersect $mdn(stem^i)$. We illustrate this difficulty in figure 2. Consider the calls P and Q, if $mdn(right(Q) \cap stem^{*i}) \leq mdn(right(P))$ then $Q \cap stem^{*i} \subseteq P \cap stem^{*i}$ and P should be preempted. Otherwise it should not be preempted.

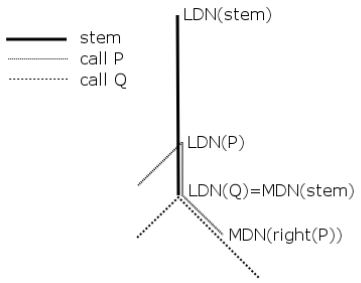

Figure 2: An example illustrating the difficulty of determining containment relations of the intersections of the calls with the stem in an online setting

To overcome this problem we make a further distinction between the calls. After a new call P has been assigned to a subsequence σ^i and the stem structure has been updated, we classify it as *determined* or *undetermined* depending on its relation to the stem structure. If $P \cap mdn(stem^i) = \emptyset$ we classify it as a determined call, otherwise it is an undetermined call. We denote by $\mathcal{D}$ the set of determined calls and by $\mathcal{U}$ the set of undetermined calls. Note that each call is classified only upon arrival (following the update of the stem structure). We do not move calls from one set to the other. If P is classified as determined (i.e. $P \cap mdn(stem^i) = \emptyset$) then $P \cap (stem^{*i} \setminus stem^i) = \emptyset$. Hence, the intersection of each determined call is determined upon arrival. In contrast, the intersection of each undetermined call with its stem may change as further calls arrive.

3.2.1 Processing the determined calls

The procedure *Determined* described in figure 3 processes the determined calls by reducing the problem to a line network. It is applied to a call in $P \in \sigma^i \cap \mathcal{D}$ after the stem structure has been updated and P has been assigned to subsequence i.

Recall that *Line* is a 2-congested algorithm for maximizing edge disjoint paths on a line [1]. It is 2 competitive against a 1-congested optimal selection. To use this algorithm we reduce vertex disjointness to edge disjointness on a line by splitting each vertex into two vertices connected by an edge.

Lemma 3.5 *For all σ and i, the maximal congestion created by Determined($\sigma^i \cap \mathcal{D}$) on T is 2.*

Procedure: *Determined*
  for each incoming call $P \in \sigma^i \cap \mathcal{D}$
    Process $P \cap stem^i$ with *Line*
    Accept P if $P \cap stem^i$ was accepted by *Line*
    Preempt calls which were preempted by *Line*
  end for

Figure 3: Algorithm for processing determined calls

Proof: First note that since the intersection of the determined calls with $stem^{*i}$ is known at the time of their arrival, $Determined(\sigma^i \cap \mathcal{D}) \cap stem^i = Line((\sigma^i \cap \mathcal{D}) \cap stem^{*i})$. Since, *Line* is a 2-congested algorithm the above equality implies a bound of 2 on the maximal congestion created by $Determined(\sigma^i \cap \mathcal{D})$ on $stem^i$. By corollary 3.4 it is also a bound on the congestion in T. □

Lemma 3.6 *For all σ and i, Determined is 2 competitive on $\sigma^i \cap \mathcal{D}$ against a 1-congested optimal selection.*

Proof: To see this consider the following,

$$|OPT^{(1)}_{\sigma^i \cap \mathcal{D}}| \leq |OPT^{(1)}_{(\sigma^i \cap \mathcal{D}) \cap stem^i}| \leq 2|Line((\sigma^i \cap \mathcal{D}) \cap stem^i)| = 2|Determined(\sigma^i \cap \mathcal{D})|$$

The first inequality holds because all calls in σ^i intersect $stem^i$. The second inequality is the competitiveness of *Line* and the last equality follows because *Determined* accepts the calls which were accepted by *Line*. □

3.2.2 Processing the undetermined calls

The undetermined calls will be processed by an on-line algorithm *UnDetermined* which follows the behavior of the optimal off-line algorithm for interval scheduling also called the activity-selection problem (see [11] chapter 17). In the off-line setting, optimal maximization of disjoint calls on a line can be achieved as follows. Sort the calls in ascending order by the depth of their maximal depth nodes. Accept the first call, discard all calls which intersect it and repeat for the remaining calls.

In the on-line setting the stem provides the line structure. The ordering of the calls is limited to lower bounds given by the current $mdn(stem^i)$. When an undetermined call P arrives, we can only say that $mdn(P \cap stem^{*i}) \geq mdn(stem^i)$. The on-line algorithm resolves this uncertainty by relaxing the congestion limitation for calls whose intersection with the final stem in still undetermined. We show that keeping three options is enough to ensure the correct selection.

The algorithm keeps the accepted calls in two sets, the *fixed* and the *unfixed* calls. All of the calls in $unfixed$ intersect $mdn(stem^i)$ and none of the calls in $fixed$ do. When an undetermined call is accepted it is always added to $unfixed$, and by definition it intersects $mdn(stem^i)$. As further calls arrive $mdn(stem^i)$ may change. Once, an $unfixed$ call no longer intersects $mdn(stem^i)$, such a call will either be moved to $fixed$ or it will

be preempted. If a call is added to $fixed$ it remains there and will not be preempted by $UnDetermined$.

Procedure $UnDetermined$ described in figure 4 is applied to a call in $P \in \sigma^i \cap \mathcal{U}$ after the stem structure and specifically $stem^i$ have been updated and P has been assigned to subsequence i.

Procedure: $UnDetermined$
Initialize:
$unfixed \leftarrow \emptyset$
$fixed \leftarrow \emptyset$
for each incoming call $P \in \sigma^i \cap \mathcal{U}$
(following the update of the stem and assignment of P to σ^i)
(1) **if** $P \cap fixed \neq \emptyset$ reject P
(2) **elseif** $\exists Q \in unfixed, Q \cap mdn(stem^i) = \emptyset$
(note: this happens only if P extended $stem^i$)
Let $F \in unfixed$ such that $depth(mdn(F \cap stem^i))$ is minimal,
$fixed \leftarrow fixed \cup F$
Preempt all calls in $unfixed$
$unfixed \leftarrow P$
(P intersects $stem^i$ only at $mdn(stem^i)$ and hence does not intersect F)
(3) **elseif** $P \cup unfixed$ creates a congestion of 4 on $mdn(stem^i)$
Let $Q_1 \in unfixed \cup P$ such that $ldn(Q_1) = mdn(stem^i)$
and let $Q_2, Q_3 \in unfixed \cup P$ such that
$depth(mdn(Q_2 \cap left(Q_1))), depth(mdn(Q_3 \cap right(Q_1)))$ are minimal
(breaking ties arbitrarily)
$unfixed \leftarrow \{Q_1, Q_2, Q_3\}$ (possibly $Q_2 = Q_3$)
Discard the remaining call (or calls)
(4) **else**
$unfixed \leftarrow unfixed \cup P$
end if
end for

Figure 4: Algorithm for processing undetermined calls

Lemma 3.7 *For all σ and i, the maximal congestion created by $UnDetermined(\sigma^i \cap \mathcal{U})$ on T is 3.*

Proof: We claim that when a call is accepted it never creates a congestion of more than 3. First we note that any accepted call passed step (1), meaning it does not intersect calls in $fixed$. If P is accepted in step (2), any call $Q \in unfixed$ which intersects it is preempted. If it is accepted in step (3), it intersects at most two other calls. If it is accepted in step (4) it must have passed step (3) meaning it does not create a congestion greater than 3. □

Lemma 3.8 *If a call $P \in unfixed$ is discarded in step (3) there is another call $Q \in unfixed$ which is not discarded, such that $mdn(Q \cap stem^{*i}) \leq mdn(P \cap stem^{*i})$.*

Proof: Let Q_1 be the call from step (3) such that $ldn(Q_1) = mdn(stem^i)$. If $mdn(Q_1 \cap stem^{*i}) \leq mdn(P \cap stem^{*i})$ we are done. If $mdn(Q_1 \cap stem^{*i}) > ldn(Q_1)$ then assume without loss of generality that the monotonic path $Q_1 \cap stem^{*i} \subseteq left(Q_1)$. Consider the call Q_2 from step (3) whose intersection with $left(Q_1)$ had a minimal maximal depth node. Since $mdn(P \cap stem^{*i}) < mdn(Q_1 \cap stem^{*i})$ assuming $mdn(Q_2 \cap stem^{*i}) > mdn(P \cap stem^{*i})$ is a contradiction to the minimality. □

Lemma 3.9 *Let $F_1, \ldots, F_m$ be the calls in $fixed$ added in that order. Let C be the set of calls which arrived between F_j and F_{j+1} for some $j = 0, \ldots, m-1$ and did not intersect F_j (where $F_0 = \emptyset$). Then $\forall Q \in C$ we have $mdn(F_{j+1} \cap stem^{*i}) \leq mdn(Q \cap stem^{*i})$.*

Proof: We consider two kinds of calls in C, when F_{j+1} is added to $fixed$ some of the calls in C are in $unfixed$ and the rest have been discarded.

We first consider the calls in $unfixed$. Since $stem^i \subseteq stem^{*i}$ for all calls $Q \in \sigma^i$ then $mdn(Q \cap stem^i) \leq mdn(Q \cap stem^{*i})$. When F_{j+1} is added to $fixed$ its intersection with $stem^{*i}$ is already determined. In particular $mdn(F_{j+1} \cap stem^i) = mdn(F_{j+1} \cap stem^{*i})$. Thus if Q is minimal in $\{mdn(Q \cap stem^i) | Q \in unfixed\}$ it is also minimal in $\{mdn(Q \cap stem^{*i}) | Q \in unfixed\}$.

The calls in $C \setminus unfixed$ have all been discarded at step (3). By lemma 3.8 every time we discard one of these calls there is a call in $unfixed$ whose intersection with $stem^{*i}$ has a maximal depth node whose depth is not greater then the maximal depth node of the discarded call. Thus the minimum of $\{mdn(Q \cap stem^{*i}) | Q \in unfixed\}$ is a lower bound on $\{mdn(Q \cap stem^{*i}) | Q \in C \setminus unfixed\}$. □

Lemma 3.10 *Each call Q rejected by the algorithm $UnDetermined$ intersects $fixed$ (and hence $fixed^*$).*

Proof: First note that calls in $fixed$ are never preempted. If Q is rejected in step (1) it intersects some call in $fixed$. If a call is discarded in step (2) it intersects the call which is added to $fixed$ at that step. Let $j = 0, \ldots m-1$ and assume by contradiction that a call Q which is rejected at step (3) lies between fixed calls F_j (if it exists) and F_{j+1} and does not intersect them. Q must have arrived after F_j (if it exists) since otherwise F_j would have been determined and before F_{j+1} or else Q would have been determined. If Q is disjoint of F_{j+1} in particular $mdn(F_{j+1} \cap stem^{*i}) > mdn(Q \cap stem^{*i})$ which is a contradiction to lemma 3.9. □

Algorithm $UnDetermined$ is very similar to the optimal algorithm for interval scheduling in that it accepts at any stage a disjoint interval that ends first. For completeness we prove that $UnDetermined$ is also 1-competitive.

Lemma 3.11 *For all σ and i, $UnDetermined$ is 1-competitive on $\sigma^i \cap \mathcal{U}$ against a 1-congested optimal selection.*

Proof: Let $F_1, \ldots, F_m$ be the calls in $fixed$ added in that order. We will show by induction on $j = 1, \ldots, m$ that there exists an optimal solution for σ^{*i} which includes $\{F_l\}_{l \leq j}$ as the first j calls.

Induction base: Let OPT be some optimal solution, we will show that there is an optimal solution whose first call is the first call the on-line algorithm added to $fixed$, F_1. If

$F_1 \notin OPT$ let Q_1 be the call in OPT whose intersection with $stem^{*i}$ has the least maximal depth node.

We will show $mdn(F_1 \cap stem^{*i}) \leq mdn(Q_1 \cap stem^{*i})$. Assume the opposite, then Q_1 must have arrived before F_1 was added to $fixed$ because otherwise it would have been determined. However this is a contradiction to lemma 3.9.

Let $OPT' \leftarrow (OPT \setminus Q_1) \cup F_1$. Because $mdn(F_1 \cap stem^{*i}) \leq mdn(Q_1 \cap stem^{*i})$, the calls in OPT' are disjoint, and since OPT' has the same number of calls as OPT, it is also optimal.

Induction step: Assume the last call added to $fixed$ was F_j, Let $\sigma^{*i}_{(j+1)}$ denote the set of all calls $P \in \sigma^{*i}$ such that $ldn(P) > mdn(F_j \cap stem^{*i})$. By induction assumption there is an optimal solution OPT for σ^{*i} which includes $\bigcup\{F_l\}_{l \leq j}$ as its first j calls. We claim that $OPT_{(j+1)} = OPT \setminus \{F_l\}_{l \leq j}$ is an optimal solution for $\sigma^{*i}_{(j+1)}$ since, if we could find a solution $OPT'_{(j+1)}$ to $\sigma^{*i}_{(j+1)}$ with more calls than $OPT_{(j+1)}$, $OPT'_{(j+1)} \bigcup\{F_l\}_{l \leq j}$ would yield a solution to σ^{*i} with more calls than OPT.

We will show that there is an optimal solution for $\sigma^{*i}_{(j+1)}$ which includes, F_{j+1} as its first call. If $F_{j+1} \notin OPT_{(j+1)}$ let Q_{j+1} be the call in $OPT_{(j+1)}$ whose intersection with $stem^{*i}$ has the least maximal depth node.

We claim that $mdn(F_{j+1} \cap stem^{*i}) \leq mdn(Q_{j+1} \cap stem^{*i})$. Assume the opposite, again note that Q_{j+1} must have arrived before F_{j+1} was added to $fixed$ because otherwise it would have been determined. Q_{j+1} was not rejected in step (1) because $ldn(Q_{j+1}) > mdn(F_j \cap stem^{*i})$ (recall that $Q_{j+1} \in \sigma^{*i}_{(j+1)}$). Nevertheless F_{j+1} is the call that was added to $fixed$, which leads to a contradiction to lemma 3.9. This concludes the induction.

Now that we have an optimal solution whose first m calls are $\{F_l\}_{l \leq m}$ all that remains is to show that there are no more calls in this optimal solution. According to lemma 3.10 all the calls which were not accepted intersect a call in $\{F_l\}_{l \leq m}$ Therefore, the algorithm produces an optimal solution. □

The following lemma will be used in the next subsection

Lemma 3.12 *For all i the instance of UnDetermined processing $\sigma^i \cap \mathcal{U}$ has a call $Q \in unfixed$ such that $ldn(Q) = mdn(stem^i)$.*

Proof: By induction on the calls in σ^i. For the base case we note that the first call to arrive is accepted and its least depth node is the maximal depth node of the stem. We proceed to the inductive step. By inductive assumption we have a call $Q \in unfixed$ such that $ldn(Q) = mdn(stem^i)$. Denote the incoming call by P and the revised stem after its arrival by $stem'^i$. If the incoming call P invokes step (2) at the end of it we accept P and $ldn(P) = mdn(stem'^i)$. When we go to step (3) we keep a call Q_1 such that $ldn(Q_1) = mdn(stem'^i)$. In step (4) P is accepted and no calls are preempted, if $stem^i = stem'^i$ then $ldn(Q) = mdn(stem'^i)$, otherwise the stem has changed because of P and then $ldn(P) = mdn(stem'^i)$. □

3.2.3 Processing the calls in σ^i

The procedure *SubSeq* (figure 5) is applied to a call P after it has been assigned to subsequence i and $stem^i$ has been updated.

Procedure: *SubSeq*
  for each incoming call $P \in \sigma^i$
    if $(P \cap mdn(stem^i) = \emptyset)$
      Process P with *Determined*
    else
      Process P with *UnDetermined*
    end if
    Accept P if it was accepted by the algorithm it was assigned to
    Preempt calls which were preempted by that algorithm
  end for

Figure 5: Algorithm for processing calls in σ^i

Lemma 3.13 *For all σ and i, the maximal congestion created by $SubSeq(\sigma^i)$ on T is 4.*

Proof: According to Lemma 3.5 the maximal congestion created by $Determined(\sigma^i \cap \mathcal{D})$ is 2 and following Lemma 3.7 the maximal congestion created by $UnDetermined(\sigma^i \cap \mathcal{U})$ is 3. *SubSeq* accepts the union of the selections, achieving a selection with a maximal congestion which is no greater than the sum of the maximal congestion created by each selection. In fact this upper bound is never attained. By lemma 3.12 we know that there is an undetermined call $Q \in unfixed$ whose least depth node is the maximal depth node of the stem. Hence, a congestion of 3 can be created by the undetermined calls only on nodes whose depth is no smaller than the maximal depth node of the stem. On the other hand all determined calls lie on nodes with a smaller depth. □

Theorem 3.14 *For all σ and i, SubSeq is 2 competitive on σ^i against a 1-congested optimal solution.*

Proof: For all σ and i,

$$\begin{aligned} |OPT^{(1)}_{\sigma^i}| &= |OPT^{(1)}_{(\sigma^i \cap \mathcal{D}) \cup (\sigma^i \cap \mathcal{U})}| \\ &\leq |OPT^{(1)}_{\sigma^i \cap \mathcal{D}}| + |OPT^{(1)}_{\sigma^i \cap \mathcal{U}}| \\ &\leq 2|SubSeq(\sigma^i) \cap \mathcal{D})| + |SubSeq(\sigma^i) \cap \mathcal{U}| \leq 2|SubSeq(\sigma^i)| \ . \end{aligned}$$

Note that *SubSeq* processes determined and undetermined calls independently accepting the union of the selections. Thus, $SubSeq(\sigma^i) \cap \mathcal{D} = Determined(\sigma^i \cap \mathcal{D})$ and $SubSeq(\sigma^i) \cap \mathcal{U} = UnDetermined(\sigma^i \cap \mathcal{U})$ so, the second inequality follows from lemmas 3.6 and 3.11. □

3.3 Combining the calls from subsequences

So far we have shown an algorithm for processing calls in each subsequence, accepting the union of the selections made on each subsequence will result in a globally competitive algorithm. However, since calls in distinct subsequences may intersect, locally bounding the congestion created by σ^i on $stem^i$ does not ensure a global bound.

To attain the global bound we introduce the procedure *Global*. This procedure uses procedure *StemStructure* to partition σ and maintain the stem structure. It then simulates *SubSeq* on each subsequence σ^i. *Global* follows the decisions made by each instance of *SubSeq* but preempts any calls which intersect more than one stem. Note that a call may intersect two stems at the moment of its arrival, or it may come to intersect two stems after it has been accepted, when a new stem is created. In both cases these calls are discarded by *Global*, however the corresponding *SubSeq* algorithm is unaware of these changes and continues to behave as if the calls are there.

An incoming request P is handled by Procedure *Global* described in figure 6.

Procedure: *Global*
  for each incoming call $P \in \sigma$
    Use procedure *StemStructure* to add P to a subsequence σ^i and update $stem^i$
    Simulate *SubSeq* on σ^i and accept / discard calls
    which were accepted / discarded by *SubSeq*
    Preempt any calls which intersect two stems (do not update simulations)
  end for

Figure 6: Global algorithm

Theorem 3.15 *For all σ, the maximal congestion created by Global(σ) on T is 4.*

Proof: By claim 3.13 the congestion created by the calls in each subsequence is bounded by 4. By discarding calls we may only reduce this congestion. Discarding calls that intersect two stems ensures there is no intersection between calls in different subsequences. If two calls P and Q from distinct subsequences intersect then the least depth node of one call, say P is contained in the intersection. Thus Q intersects its own stem and $ldn(P)$ which belongs to another stem. □

Lemma 3.16 *For all i, if $P \in \sigma^i$ such that $ldn(P) = mdn(stem^i)$ then P does not intersect two stems.*

Proof: Assume P intersects $stem^j$ for $j \neq i$. Since $ldn(P) = mdn(stem^i)$ its intersection with $stem^j$ lies in the subtree rooted at $ldn(P)$, but by properties of the stem structure we know that $mdn(stem^i)$ is a leaf of S, the tree that connects the least depth nodes of the calls in σ and r. □

Lemma 3.17 *Let k be the number of subsequences, then $k \leq |Global(\sigma) \cap \mathcal{U}|$.*

Proof: By lemma 3.12 for all i, *UnDetermined* has a call $P \in unfixed$ such that $ldn(P) = mdn(stem^i)$. This call is accepted by *SubSeq* and by lemma 3.16 it is not preempted by *Global*. Thus, at least one call from each subsequence σ^i is accepted, since there are k of these, *Global* accepts at least k undetermined calls. □

Lemma 3.18 *The procedure Global discards at most* $3(k-1)$ *calls which were accepted by the instances of SubSeq, of which at most* $2(k-1)$ *are determined calls and* $k-1$ *are undetermined calls. Specifically,*

$$\sum_{i=1}^{k} |SubSeq(\sigma^i) \cap \mathcal{D}| \leq |Global(\sigma) \cap \mathcal{D}| + 2(k-1)$$

and

$$\sum_{i=1}^{k} |SubSeq(\sigma^i) \cap \mathcal{U}| \leq |Global(\sigma) \cap \mathcal{U}| + (k-1).$$

Proof: For all $i = 2, \ldots, k$, $stem^i$ has a root node v_i. The set $\{v_i\}_{i=2}^{k}$ includes at most $k-1$ distinct vertices. We recall that a stem's root node is fixed at the moment of its creation and does not change.

Any call accepted by *SubSeq* running on σ^i, which is preempted by *Global* must intersect some root node $v_j \in stem^i$ such that $stem^i$ is the parent of $stem^j$. The selection made by *SubSeq* on σ^i may include at most two determined calls and one undetermined call that intersect some root node $v_j \in stem^i$.

Thus the total amount of calls which are discarded is bounded by $3(k-1)$ of which at most $2(k-1)$ are determined and $k-1$ are undetermined. □

Theorem 3.19 *For all* σ*, Global is 6 competitive on* σ *against a* 1*-congested optimal selection (recall that Global is* 4*-congested).*

Proof: For all σ,

$$\begin{aligned}
|OPT_{\sigma}^{(1)}| &= |OPT_{\cup_{i=1}^{k}\sigma^i}^{(1)}| \leq \sum_{i=1}^{k} |OPT_{\sigma^i}^{(1)}| \\
&\leq \sum_{i=1}^{k} 2|SubSeq(\sigma^i) \cap \mathcal{D}| + |SubSeq(\sigma^i) \cap \mathcal{U}| \\
&\leq 2|Global(\sigma) \cap \mathcal{D}| + 4(k-1) + |Global(\sigma) \cap \mathcal{U}| + (k-1) \\
&\leq 2|Global(\sigma) \cap \mathcal{D}| + 6|Global(\sigma) \cap \mathcal{U}| \leq 6|Global(\sigma)|.
\end{aligned}$$

The second inequality follows from the proof of claim 3.14. The third inequality is due to lemma 3.18. The fourth follows from lemma 3.17 and the last is because $\sigma = \mathcal{D} \cup \mathcal{U}$. □

4 A constant competitive randomized algorithm for disjoint paths

In this section we present a 1-congested, randomized 24-competitive algorithm. We show that the calls accepted by *Global* can be assigned in an online manner into a small number of1-congested sets. The randomized algorithm randomly chooses one of these sets and simulates *Global*. It accepts only the calls which are assigned to the chosen set and discards the rest.

Definition 4.1 *Let $\mathcal{A}$ be an on-line algorithm and let C denote a set of calls which is maintained by $\mathcal{A}$. An* on-line *d-coloring of C is an on-line assignment $\chi : C \mapsto \{1, \ldots, d\}$. When a call P is added to C the on-line coloring assigns it to some color class $\chi(P) \in \{1, \ldots, d\}$. The assignment is made at the time of P's arrival and may not be changed. The coloring is* valid *if for any two calls $P \neq Q \in C, \chi(P) = \chi(Q) \Rightarrow P \cap Q = \emptyset$.*

Lemma 4.2 *If an on-line algorithm maintains a set of calls C, such that after a call P has been added to C, it intersects at most $d-1$ other calls in C, then there exists a valid on-line d-coloring for C.*

Proof: We prove this by induction on the calls added to C. The base case is trivial since $C = \emptyset$. By the inductive assumption up until P's arrival the calls in C have been assigned to valid color classes $\{C_j\}_{j=1}^{d}$. After P is accepted it intersect at most $d-1$ other calls in C so we can set $\chi(P) = min\{j | P \cap C_j = \emptyset\}$ to get a valid coloring of $C \cup P$. □

Lemma 4.3 *There exists a valid on-line 6-coloring for the calls maintained by Global.*

Proof: We color the calls in D and in U independently.

If $P \in \sigma^i$ was accepted as a determined call its intersection with $stem^i$ must have been accepted by *Line*. Since *Line* is a 2-congested algorithm and never accepts a call which contains another, P may intersect at most two other calls on $stem^i$. One with a smaller depth least depth node and one with a larger depth maximal depth node. By corollary 3.4 the same holds anywhere on T. Hence, P intersects at most two other calls in $\sigma^i \cap \mathcal{D}$. Since P was not preempted by *Global* it does not intersect calls from other subsequences. Thus lemma 4.2 can be applied to get an on-line 3-coloring for $Global(\sigma) \cap \mathcal{D}$.

If P was accepted as an undetermined call, following lemma 3.7 it intersects at most two other calls in $\sigma^i \cap \mathcal{U}$. The calls in $fixed$ are disjoint and in $unfixed$ there are at most three calls. Again, since P was not preempted by *Global* it does not intersect calls from other subsequences. Thus, lemma 4.2 provides an on-line 3-coloring for $Global(\sigma) \cap \mathcal{U}$.

Using the above 3-colorings we assign $Global(\sigma) \cap \mathcal{D}$ to color classes $\{1, 2, 3\}$ and $Global(\sigma) \cap \mathcal{U}$ to color classes $\{4, 5, 6\}$ to get a valid on-line 6-coloring for $Global(\sigma)$. □

Using the on-line coloring from lemma 4.3 we construct a randomized 1-congested, algorithm *Rand* as described in figure 7.

Theorem 4.4 *For all σ, the maximal congestion created by $Rand(\sigma)$ on T is 1.*

Proof: *Rand* only accepts calls from a single color, and each color class is comprised of disjoint calls. □

Theorem 4.5 *Rand is a 24 competitive algorithm.*

Proof: For all σ,

$$\begin{aligned} E(|Rand(\sigma)|) &= \frac{1}{12}\sum_{j=1}^{3} |(Global(\sigma) \cap \mathcal{D}) \cap C_j| + \frac{1}{4}\sum_{j=4}^{6} |(Global(\sigma) \cap \mathcal{U}) \cap C_j| \\ &= \frac{1}{24}(2|Global(\sigma) \cap \mathcal{D}| + 6|Global(\sigma) \cap \mathcal{U}|) \geq \frac{1}{24}|\mathcal{OPT}_{\sigma}^{(1)}| \end{aligned}$$

Procedure: *Rand*
Make a random selection $i \in \{1, \ldots, 6\}$ with probability $Pr[i] = \frac{1}{12}$ for $i \in \{1, 2, 3\}$ and $Pr[i] = \frac{1}{4}$ for $i \in \{4, 5, 6\}$
for each incoming call $P \in \sigma$
Simulate *Global* on P
if P is accepted by *Global* and $\chi(P) = i$,
Accept P
else
Reject P
end if
Preempt calls which were preempted by *Global*
end for

Figure 7: Disjoint path algorithm

where $C_j = \{P|\chi(P) = j\}$. The first equality is the expansion of the expectation. The second equality holds since $\bigcup_{j=1}^{3} C_j = |Global(\sigma) \cap \mathcal{D}|$ and $\bigcup_{j=4}^{6} C_j = |Global(\sigma) \cap \mathcal{U}|$. The last inequality follows from the proof of theorem 3.19. □

4.1 A constant competitive randomized algorithm for congestion b

In sections 3 and 4 we considered the vertex disjoint paths problem. In this section we extend the setting and allow a bounded maximal congestion of $b > 1$. Following [1] we use the general method of [5] for benefit problems. We also take advantage of its adaptation to handle preemption as presented in [1].

The framework of [1, 5] considers an on-line benefit problem where the benefit is gained by allocating items into b independent "bins". The method there shows how to use a preemptive deterministic (or randomized) ρ-competitive algorithm for a single bin to construct a preemptive deterministic (or randomized) $(\rho + 1)$-competitive algorithm for b bins. The method is based on applying b copies of the single bin algorithm where each request is pipelined through the copies (in a fixed order) until it is accepted. A request is rejected if no copy would accept it.

Claim 4.6 *Any feasible solution to the vertex b-congested paths problem on a tree T can be partitioned into no more than b independent feasible solutions for the vertex disjoint problem on T.*

Proof: Consider a set of paths C such that the maximal congestion created by C on T is b. Order all the paths $P \in C$ by non-decreasing order of $ldn(P)$. We color the paths inductively following the above depth order such that all paths in the same color class are disjoint (and hence feasible solutions for the vertex disjoint problem). Given a path P all the paths which intersect it and are already colored must intersect P at $ldn(P)$. Since there are at most $b - 1$ such paths there is a free color to choose for P. □

Theorem 4.7 *For all $b > 1$, There exists a b-congested, randomized, preemptive 25-competitive algorithm for the maximal vertex b-congested paths problem on trees. For $b \geq 4$ there is a deterministic constant competitive algorithm.*

Proof: We start with the randomized algorithm. By claim 4.6 a feasible solution for the b-congested problem can be partitioned into b independent feasible solutions of the disjoint paths problem. Moreover, the union of b selections for the disjoint paths problem is a feasible solution for the b-congested problem. Hence, the b-congested problem is an instance of allocation of items into bins. Using the 24 competitive *Rand* as the algorithm for a single bin we get a 25 competitive algorithm for b bins.

Next we show the deterministic algorithm. We note that by claim 4.6 the optimal b-congested solution is at most b times the optimal solution for 1-congested solution. Hence for $b = 4$ we immediately observe that *Global* is a deterministic 24 competitive algorithm. For b which is divisible by 4 we can partition b into $b/4$ bins of size 4 and get a 25 competitive algorithm as described above. If b is not divisible by 4 we round it down to a b which is divisible by 4 and use it. By that we lose a factor of at most 2 since by 4.6 the optimal x-congested solution is at least x/y of the optimal y-congested solution for any $x \leq y$. □

References

[1] R. Adler and Y. Azar. Beating the logarithmic lower bound: randomized preemptive disjoint paths and call control algorithms. In *Proc. of the 10th ACM-SIAM Symposium on Discrete Algorithms*, pages 1–10, 1999.

[2] N. Alon, U. Arad and Y. Azar. Independent sets in hypergraphs with applications to routing via fixed paths. In *Proc. 2nd Workshop on Approximation Algorithms for Combinatorial Optimization Problems (APPROX)*, pages 16-27, 1999.

[3] M. Andrews, J. Chuzhoy, S. Khanna, and L. Zhang. Hardness of the undirected edge-disjoint paths problem with congestion. In *Proceedings 46th Annual IEEE Symposium on Foundations of Computer Science*, pages 226–244, 2005.

[4] J. Aspnes, Y. Azar, A. Fiat, S. Plotkin, and O. Waarts. On-line routing of virtual circuits with applications to load balancing and machine scheduling. *Journal of the ACM*, 44(3):486–504, 1997. Also in *Proc. 25th ACM STOC*, 1993, pp. 623-631.

[5] B. Awerbuch, Y. Azar, A. Fiat, S. Leonardi, and A. Rosen. On-line competitive algorithms for call admission in optical networks. In *Proc. 4th Annual European Symposium on Algorithms*, pages 431–444, 1996.

[6] B. Awerbuch, Y. Azar, and S. Plotkin. Throughput-competitive online routing. In *34th IEEE Symposium on Foundations of Computer Science*, pages 32–40, 1993.

[7] B. Awerbuch, Y. Bartal, A. Fiat, and A. Rosén. Competitive non-preemptive call control. In *Proc. of 5th ACM-SIAM Symposium on Discrete Algorithms*, pages 312–320, 1994.

[8] B. Awerbuch, R. Gawlick, T. Leighton, and Y. Rabani. On-line admission control and circuit routing for high performance computation and communication. In *Proc. 35th IEEE Symp. on Found. of Comp. Science*, pages 412–423, 1994.

[9] Y. Bartal, A. Fiat, and S. Leonardi. Lower bounds for on-line graph problems with application to on-line circuit and optical routing. In *Proc. 28th ACM Symp. on Theory of Computing*, pages 531–540, 1996.

[10] A. Borodin and R. El-Yaniv. *Online Computation and Competitive Analysis*. Cambridge University Press, 1998.

[11] T.T. Cormen, C.E. Leiserson, and R.L. Rivest. Introduction to Algorithms. MIT Press, 1990.

[12] J. Garay, I. Gopal, S. Kutten, Y. Mansour, and M. Yung. Efficient on-line call control algorithms. *Journal of Algorithms*, 23:180–194, 1997. Also in *Proc. 2'nd Annual Israel Conference on Theory of Computing and Systems*, 1993.

[13] N. Garg, V.V. Vazirani, and M. Yannakakis. Primal-Dual Approximation Algorithms for Integral Flow and Multicut in Trees. In *ALGORITHMICA*, 18:3–20, 1997.

[14] J. Kleinberg and E. Tardos. Disjoint paths in densely embedded graphs. In *Proc. 36th IEEE Symp. on Found. of Comp. Science*, pages 52–61, 1995.

[15] S. Leonardi. On-line network routing. In A. Fiat and G. Woeginger, editors, *Online Algorithms - The State of the Art*, chapter 11, pages 242–267. Springer, 1998.

[16] S. Leonardi, A. Marchetti-Spaccamela, A. Presciutti, and A. Rosén. On-line randomized call control revisited. In *Proc. 9th ACM-SIAM Symp. on Discrete Algorithms*, pages 323–332, 1998.

[17] R. J. Lipton and A. Tomkins. Online interval scheduling. In *Proc. of the 5th ACM-SIAM Symposium on Discrete Algorithms*, pages 302–311, 1994.

[18] P. Raghavan and C. D. Thompson. Randomized rounding: a technique for provably good algorithms and algorithmic proofs. *Combinatorica*, 7(4):365–374, 1987.