An Evaluation of In-Advance Bandwidth Scheduling Algorithms for Connection-Oriented Networks

Eun-Sung Jung, Yan Li, Sanjay Ranka and Sartaj Sahni
Department of Computer and Information Science and Engineering
University of Florida, Gainesville, FL 3261
{ejung,yanli,ranka,sahni}@cise.ufl.edu

Abstract

Several bandwidth management systems have been developed to reserve, in advance, dedicated connections for high-performance applications. We describe the in-advance reservation capabilities of these systems as well as the bandwidth scheduling and path computation algorithms used. An analytical and experimental evaluation of these algorithms also is provided. Our experiments indicate that for the fixed-slot problem, the minimum-hop feasible path algorithm proposed by us in [8] maximizes network utilization for large networks while the dynamic adaptive feasible path algorithm proposed in this paper does this for small networks.

1. Introduction

Dedicated connections are needed to effectively support a variety of data intensive and geographically distributed applications including data mining, data consolidation and alignment, storage, visualization and analysis [24]. More specifically, dedicated bandwidth channels are critical in these tasks to offer (i) large capacity for massive data transfer operations, and (ii) dynamically stable bandwidth for monitoring and steering operations. It is important that these channels be available when the data is or will be ready to be transferred. Thus, the ability to reserve such dedicated bandwidth connections either on-demand or in-advance is critical to both classes of operations. The importance of dedicated connection capabilities has been recognized, and several network research projects are currently underway to develop such capabilities. They include [25, 14, 15, 16, 17, 18, 19]. In addition, production networks at the national and international scale with such capabilities are being deployed by Internet2 [20] and LHCNet[21]. Such deployments are expected to be on the increase and proliferate into both shared and private network infrastructures across the globe in the coming years.

Bandwidth reservation systems operate in one of two modes: (a) In on-demand scheduling, bandwidth is reserved for a time period that begins at the current time, and (b) In in-advance scheduling, bandwidth is reserved for a time period that begins at some future time. On-demand scheduling is a special case of in-advance scheduling; the future time for each scheduling request is separated from the time at which the request is made by a time interval of zero. On-demand scheduling, which is typically supported by Multiple Protocol Label Switching (MPLS) [22] at layer 3 and by Generalized MPLS (GMPLS) [23] at layers 1 and 2, is supported in [14, 16, 17]. Algorithms for on-demand scheduling are described in [3, 5, 6, 11], for example. GanttII, OSCARS, USN and EnLIGHTened support in-advance scheduling and algorithms for in-advance scheduling are described in [1, 4, 24, 25, 26, 8], for example.

The rest of the paper is organized as follows. In Section 2, we describe the time models that have been proposed for in-advance scheduling. The various scheduling problems formulated for in-advance scheduling are stated in Section 3 and algorithms for these scheduling problems are described in Section 4. Section 5 gives the metrics used to evaluate in-advance scheduling algorithms. Since, for problems other than the fixed-slot problem, either only one algorithm has been proposed or the relative performance of the proposed algorithms is readily deduced analytically, we experiment only with the algorithms for the fixed-slot problem. The results of our experiments are given in Section 6. Finally, we conclude in Section 7.

2. Time Models

We assume that the network is represented as a graph $G = (V, E)$. Each node of this graph represents a device such as a switch for layers 1-2 and a router for layer
3; and each edge represents a link such as SONET or Ethernet. When developing an in-advance reservation system one must decide on a representation of time. The options are to either consider time as divided into equal size slots as is done in [1, 2, 4, 9, 10] or to consider time as being continuous as in [24, 25, 26, 8, 12].

The slotted time model uses an array to store link status for each time slot. For example, we may use a two-dimensional array \( b \) such that \( b[i, t] \) gives the bandwidth available on link \( l \) in slot \( t \). Although the slotted time model allows for easy access of information, it is limited as the granularity of the time slot has to be predetermined. Choosing a small slot (for finer control in scheduling), leads to a large amount of storage as well as increases the complexity of making a pass through the entire array.

In the continuous time model, the status of each link \( l \) is maintained using a time-bandwidth list (TB list) \( TBL[l] \) that is comprised of tuples of the form \((t_i, b_i)\), where \( t_i \) is a time and \( b_i \) is a bandwidth. The tuples on a TB list are in increasing order of \( t_i \). If \((t_i, b_i)\) is a tuple of \( TBL[l] \) (other than the last one), then the bandwidth available on link \( l \) from \( t_i \) to \( t_{i+1} \) is \( b_i \). When \((t_i, b_i)\) is the last tuple, a bandwidth of \( b_i \) is available from \( t_i \) to \( \infty \). For example, the TB list \( TBL[4] = [(2, 10), (9, 5), (20, 50)] \) represents the status of link 4 that has bandwidth 10 available from time 2 to time 9, bandwidth 5 from time 9 to time 20, and bandwidth 50 thereafter. Each TB list may be represented as an array linear list using dynamic array resizing as described in [7] or as a linked list.

A big advantage is that there is no need to pick a time granularity or to place a bound on the length of the look ahead period. The amount of memory required to represent a link state (i.e., the TB list) is a function of the time variation in link bandwidth availability rather than the scheduling horizon \( T \). Also, the run-time of reservation algorithms is a function of the size of the TB lists. This size depends on the number of tasks that have been scheduled. The limitations of the continuous time model include its relative complexity (linear lists are somewhat more difficult to handle than arrays) and the complexity of determining the status of a link at any given time. The latter can be done in \( O(\log I(TBL[l])) \) time using a binary search in case of an array linear list and in \( O(|TBL[l]|) \) time in case of a linked TB list.

Because of the correspondence between a slot and time, we often use the two terms interchangeably.

### 3. In-advance Scheduling Problems

In the following paragraphs, we describe the problems of interest in the context of in-advance scheduling.

**Fixed Slot:** Reserve a path with bandwidth \( b \) from the source \( s \) to the destination \( d \) from time \( t_{\text{start}} \) to time \( t_{\text{end}} \).

**Maximum Bandwidth in Slot:** Find the largest bandwidth \( b \) such that there is a bandwidth \( b \) path from the source \( s \) to the destination \( d \) from time \( t_{\text{start}} \) to time \( t_{\text{end}} \). Reserve such a path.

**Maximum Duration:** Find the maximum duration \( \tau \) such that there is a bandwidth \( b \) path from the source \( s \) to the destination \( d \) from time \( t_{\text{start}} \) to time \( t_{\text{start}} + \tau \). Reserve such a path.

**First Slot:** Find the least \( t \) for which there is a path with bandwidth \( b \) from the source \( s \) to the destination \( d \) from time \( t \) to time \( t + \tau \), where \( \tau \) is the duration for which the path is desired. Reserve such a path.

**All Slots:** Find all ranges \( r \) such that for every \( t \in r \), there is a bandwidth \( b \) path from the source \( s \) to the destination \( d \) from time \( t \) to time \( t + \tau \). Reserve such a path for a user selected \( t \) in one of the found ranges.

**All Pairs All Slots:** For every source-destination pair \((s, d)\), find all ranges \( r \) such that for every \( t \in r \), there is a bandwidth \( b \) path from \( s \) to \( d \) from time \( t \) to time \( t + \tau \).

The fixed-slot problem is referred to as the connection feasibility problem in [4]. The soonest completion problem formulated in [4] and the first available transmission period in [1] both are identical to the first-slot problem stated above. It can be shown that most of the above problems can be reduced to conducting one or more iterations of a version of the first-slot or fixed-slot problem. Because of space limitations, we focus on the latter two problems.

### 4. Path Computation Algorithms

We describe only the algorithms needed to compute the paths for the fixed slot and first set problems described in Section 3. The actual scheduling or reservation of the found path requires us to update the TB lists or the \( b \)-array entries for the links on the path as well as to signal the routers on the path at the reserved time. The former is a relatively straightforward process and the latter requires the use of specific signaling protocols.

#### 4.1. Fixed Slot

Of the problems listed in Section 3, the fixed-slot problem is the most studied. The algorithms that have been proposed are described below.

**Feasible Path (FP):** A link of the network is feasible for fixed-slot scheduling iff the available bandwidth on the link at all times in the interval \([t_{\text{start}}, t_{\text{end}}]\) is \( \geq b \). Let \( p \) be a path from the source \( s \) to the destination \( d \). \( p \) is a feasible path iff it is comprised solely of feasible links. In FP scheduling, a feasible path is reserved. Such a path may be found by performing a search (depth- or breadth-first, for example [7]) on the subgraph of \( G \), called the feasible subgraph, obtained from \( G \) by eliminating all links that are not feasible. FP scheduling is done in [4].

**Minimum Hop Feasible Path (MHFP):** The number of links on a path is its hop count. In MHFP scheduling, a
minimum-hop feasible path is reserved. Such a path may be found by performing a breadth-first search on the feasible subgraph of $G$. Notice that FP scheduling is the same as MHFP scheduling when the search method used by FP scheduling is breadth first. MHFP scheduling is used in USN and the path computation algorithm is formally stated in [8].

**Widest/Shortest Feasible Path (WSFP):** This is an adaptation of the widest-shortest method proposed in [3] for on-demand scheduling. Let $p$ be a feasible path. Let $b_{min} \geq b$ be the minimum bandwidth available on any link of $p$ at any instant in the scheduling interval $[t_{start}, t_{end}]$. In WSFP scheduling, we use the minimum-hop feasible path that has the maximum $b_{min}$ value. Ties are broken arbitrarily. Notice that WSFP scheduling is MHFP scheduling with a specified tie breaker. WSFP scheduling is suggested in [1]. A WSFP may be found by running a modified Bellman-Ford algorithm on the feasible subgraph of $G$ [3]; the weight of a link is the minimum bandwidth available on that link during the interval $[t_{start}, t_{end}]$ or by running a modified version of Dijkstra’s shortest path algorithm on this feasible subgraph [6]. In the latter case, when selecting the next shortest path, priority is given to next-path candidates with least hop count and ties are broken by using the $b_{min}$ value of the path.

**Shortest/Widest Feasible Path (SWFP):** This is a variant of WSFP scheduling that was first proposed for on-demand scheduling [11]. We select a feasible path that has maximum $b_{min}$ value. Ties are broken by favoring paths with smaller hop counts. An SWFP path may be found [6] by first running Dijkstra’s shortest path algorithm modified to find a path with maximum $b_{min}$ and then doing a breadth-first search to find a minimum-hop path with this maximum $b_{min}$ value; the breadth-first search ignores links that violate this $b_{min}$ requirement.

**Shortest Distance Feasible Path Algorithms (SDFP):** These algorithms find a shortest path (the length of a path being the sum of the weights of the links on that path) in the feasible subgraph of $G$. An SDFP path may be found using Dijkstra’s shortest path algorithm. SDFP algorithms differ in their selection of a cost metric for feasible links. SDFP-min (minimum SDFP) is an extension of the shortest distance path algorithm for on-demand scheduling [5] to the case of in-advance scheduling. The weight of a feasible link is the reciprocal of the minimum bandwidth available on that link during the scheduling interval $[t_{start}, t_{end}]$. In SDFP-avg (average SDFP), the weight of a feasible link is the reciprocal of the average.

**Dynamic Alternative Feasible Path (DAFP):** Again, this is an adaptation of the dynamic alternative path algorithm originally proposed for on-demand scheduling [6]. Let $h$ be the number of hops in the MHFP. In DAFP, we use a widest feasible path (i.e., one with maximum $b_{min}$ value) that has no more than $h + 1$ hops. Such a path may be found [6] by restricting the Bellman-Ford algorithm proposed for WSFP to use no path with more than $h + 1$ hops. We note that while DAFP guarantees to find a feasible path whenever such a path exists, the dynamic alternative path algorithm of [6] provides no such guarantee.

**OSPF Like Algorithms:** These are shortest path algorithms that work on $G$ or some subgraph of $G$ other than the feasible subgraph. They differ in how the link weights are defined and/or in how the subgraph is defined. Since these algorithms do not work on the feasible subgraph of $G$, they may generate an infeasible path and fail to schedule a request in some cases where one of the aforementioned feasible path algorithms succeed. The shortest path may be found using Dijkstra’s algorithm. In the version of OSPF-TE implemented in OSCARS [19], you remove from the network graph $G$ those links that do not have an available bandwidth that is at least $b$ at the time the scheduling request is processed (not at time $t_{start}$); link weights are as for OSPF. The shortest path in this reduced graph is found and an attempt is made to schedule the reservation request on this path.

**$k$ Dynamic Paths (kDP):** These algorithms are an extension of OSPF-like algorithms. Recognizing that an OSPF-like algorithm may fail to find a feasible path in a network that has a feasible path, kDP algorithms generate additional paths with the hope that one of the additional paths will be feasible. In a kDP algorithm, when the generated path is infeasible, we reduce the current graph by removing from it links on the generated infeasible path whose available bandwidth during the reservation interval $[t_{start}, t_{end}]$ is less than $b$. We then find the shortest $s$ to $d$ path in this reduced graph. This path computation and graph reduction process is repeated at most $k$ times. The process terminates when the first feasible path is found or when $k$ infeasible paths have been generated. kDP-OSPF and kDP-OSPF-TE are natural extensions of OSPF and OSPF-TE. kDP-LOAD is an adaptation of the algorithms used in [10] for in-advance scheduling of optical networks. In kDP-LOAD, each link is assigned a weight equal to the total load already scheduled on that link.

**$k$ Static Paths (kSP):** In this algorithm, we have up to $k$ precomputed paths between every pair of source-destination vertices. To schedule a path between $s$ and $d$, we examine, in some order, the up to $k$ precomputed paths for the pair $(s, t)$ and select the first that is feasible for the interval $[t_{start}, t_{end}]$. If none is feasible, the scheduling request is denied.

### 4.2. First Slot

Three different algorithms have been proposed for the first slot problem [4, 8, 10]. These are described below. **Slotted Sliding Window (SSW):** The sliding window first algorithm proposed in [10] for optical networks is a variation of the shortest completion algorithm proposed in [4].
Both these algorithms try the slots $t_{\text{start}}$, $t_{\text{start}} + 1$, \ldots, in this order, to find the least $t$ for which the graph $G$ has a feasible path (i.e., an $s$ to $d$ path with bandwidth $b$ for the duration from $t$ to $t + \tau$). The existence of a feasible path for any $t$ may be done using a fixed-slot algorithm such as FP that guarantees to find a feasible path whenever such a path exists or by using a kDP (as in the case of [10]) or kSP algorithm that does not provide such a guarantee.

**List Sliding Window (LSW):** This is similar to SSW except that it was developed for the continuous time model in which there is no concept of discrete time intervals (i.e., slots). For each link in the network, we define a start-time list, $ST$, that is comprised of pairs of the form $(a, b)$ with the property that for every $t \in [a, b]$, the link has bandwidth $b$ available from $t$ to $t + \tau$. Let $a_1 < a_2 < \cdots < a_q$ be the distinct $a$ values in the union of the $ST$ lists of all links. It is easy to see that the earliest time $t$ for which the network has a path from $s$ to $d$ with bandwidth $b$ from time $t$ to time $t + \tau$ is one of the $a_i$. The LSW algorithm of [8] examines the $a_i$s in the order $a_1, a_2, \cdots$ stopping at the first $a_i$ for which a feasible path is found. The search for a feasible path is done using a breadth-first search as is done by the MHFP algorithm. Some optimization is possible since the breadth-first search for $a_i$ follows that for $a_{i-1} < a_i$. Since the breadth-first search must scan the $ST$ list of each link that is traversed during the search, this scan may begin where the most recent scan of this list (from the breadth-first search for an earlier $a_i$) left off rather than from the front of the $ST$ list.

Although LSW was developed for the continuous time model, it may be used also in the slotted time model regardless of whether $TB$ lists or an array is used to represent link status.

**Extended Bellman Ford (EBF):** This algorithm for the first slot was proposed in [8]. First, we extend the notion of an $ST$ list for a link to that for a path in the natural way. Next, define $st(k, u)$ to be the union of the $ST$ lists for all paths from vertex $s$ to vertex $u$ that have at most $k$ edges on them. Clearly, $st(0, u) = \emptyset$ for $u \neq s$ and $st(0, s) = [0, \infty]$. Also, $st(1, u) = ST(s, u)$ for $u \neq s$ and $st(1, s) = st(0, s)$. For $k > 1$ (actually also for $k = 1$), we obtain the following recurrence

$$st(k, u) = st(k - 1, u) \cup \{v \mid u \text{ such that } (v, u) \text{ is an edge}\}$$

where $\cup$ and $\cap$ are list union and intersection operations. For an $n$-vertex graph, $st(n - 1, d)$ gives the start times of all paths from $s$ to $d$ that have bandwidth $b$ available for a duration $\tau$. The $a$ value of the first $(a, b)$ pair in $st(n - 1, d)$ gives the desired first slot.

The Bellman-Ford algorithm [7] may be extended to compute $st(n - 1, d)$. The extension merely works with $st$ lists rather than with scalars and is described in [8].

### Table 1. Algorithm time complexities

<table>
<thead>
<tr>
<th>Prob.</th>
<th>Algorithm</th>
<th>SlottedArray</th>
<th>Continuous</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fixed Slot</td>
<td>FP</td>
<td>$O(eT)$</td>
<td>$O(eT)$</td>
</tr>
<tr>
<td>MHFP</td>
<td>$O(eT)$</td>
<td>$O(eT + n \log n)$</td>
<td>$O(e + L)$</td>
</tr>
<tr>
<td>WSFF/WSWF, SDFP/DAFP</td>
<td>$O(e + n(\tau + \log n))$</td>
<td>$O(e + L + n \log n)$</td>
<td>$O(e + L + n \log n)$</td>
</tr>
<tr>
<td>OSPF</td>
<td>$O(k(e + \min{kn, e}T + \log n))$</td>
<td>$O(k(e + L + n \log n) + L + n \log n)$</td>
<td>$O(k(e + L + n \log n) + L + n \log n)$</td>
</tr>
<tr>
<td>kDP</td>
<td>$O(\min{kn, e}T + \log n)$</td>
<td>$O(L + kn)$</td>
<td></td>
</tr>
<tr>
<td>kSP</td>
<td>$O((q + T)\epsilon)$</td>
<td>$O(qe + L)$</td>
<td></td>
</tr>
<tr>
<td>First Slot</td>
<td>SSW-MHFP</td>
<td>$O(\log n)$</td>
<td>$O(\log n)$</td>
</tr>
<tr>
<td>LSW</td>
<td>$O(n + eT)$</td>
<td>$O(\log n)$</td>
<td></td>
</tr>
<tr>
<td>EBF</td>
<td>$O(n + eT)$</td>
<td>$O(\log n)$</td>
<td></td>
</tr>
</tbody>
</table>

$l = \text{size of longest } st \text{ list}, q = \text{number of different } a_i \text{s in the } ST \text{ lists}, L = \text{sum of lengths of } TB \text{ lists}$

### 5. Performance Metrics

In addition to the traditional metrics of space and time complexity, the effectiveness of an in-advance scheduling algorithm is critical. The space requirement should not exceed the available memory on the computer on which the bandwidth management system is to run. The time complexity influences the response time of the bandwidth management system and, in turn, determines how many reservation requests this system can process per unit time. Scheduling effectiveness is, of course, critical as revenue is generated only from tasks that are actually scheduled. The time and space requirements of the various fixed- and first-slot algorithms are given in Table 1.

There are two aspects to effectiveness—guarantees and utilization. Guarantees have to do with whether or not the scheduling algorithm provides any guarantee on its result. For example, does a fixed-slot algorithm guarantee to find a feasible path whenever such a path exists? Does a first-slot algorithm actually find the earliest feasible path? For the first slot problem, there are three algorithms—SSW-MHFP, LSW, and EBF. All 3 guarantee to find the first slot correctly. Hence, barring differences resulting from their possible implementation using different tie breakers, each is just as effective. Of course, there will be some difference in the computer-time taken to execute each algorithm. For the fixed-slot problem, all algorithms other than the OSPF-like, $k$ dynamic paths, and $k$ static paths algorithms provide a guarantee. An experimental evaluation of effectiveness is needed to compare these algorithms and is the focus of the next section.

The scheduling algorithms work in an online mode: scheduling requests are processed in the order they arrive at the bandwidth management system and a decision on whether or not to make the requested reservation made on the basis of link states at the time the reservation request is processed without regard to future requests. Hence, the link status at the time a decision is made on the current request being processed depends on decisions made in the
past. These past decisions are a function of the path computation algorithm(s) in use. Therefore, network utilization as measured by the number of accepted requests or the total network bandwidth that has been scheduled may be more using OSPF-TE that provides no guarantee than when using FP that provides a guarantee! Two metrics request acceptance (RAR) and bandwidth acceptance (BAR) ratios can be defined in the same vein as in [1]:

\[
RAR = \frac{\text{number of accepted requests}}{\text{total number of requests}}
\]

\[
BAR = \frac{\text{sum of bandwidth-duration products of accepted requests}}{\text{sum of bandwidth-duration products of all requests}}
\]

We found that the relative performance of the fixed-slot algorithms is rather insensitive to whether we use the RAR or BAR metric, we report only on the RAR results.

6. Experiments

We programmed the various fixed-slot algorithms in C++ and measured the effectiveness of each using the RAR and BAR metrics. Although we experimented with both variants (SDFP-min and SDFP-avg) of SDFP, we report only the results for SDFP-min as the results for both variants are comparable. For OSPF, we programmed the OSPF-TE variant that is used in OSCARS [19]. The kDP variant tested by us is kDP-LOAD with \( k = 4 \). We used this variant as it is the variant used in EnLIGHTened [15]. Finally, for kSP, we set \( k \) to 4 and used 4 shortest paths.

For test networks, we used the 19-node MCI network and the 16-node cluster network of [6], the 11-node network of [1], the Abilene network [13], and several randomly generated topologies. We only present results for the MCI network due to space limitations. The bandwidth of each link in the network of [1] is 100Mbps. The random networks we tried had 200, 400, 600, 800, and 1000 nodes and the out-degree of each node was randomly selected to be between 3 and 7. To ensure network connectivity, the random network has bidirectional links between nodes \( i \) and \( i + 1 \) for every \( 1 \leq i < n \), where \( n \) is the number of nodes. The bandwidth of each link in a randomly generated network was randomly selected from the set OC1, OC3, OC12, OC24, OC48, OC96, OC92, 1G Ethernet and 10G Ethernet.

We generated a synthetic set of reservation requests. Each request is described by the 6-tuple (source node, destination node, time at which the request is scheduled, requested start time, duration, bandwidth). The source and destination nodes for each request were randomly selected using a uniform random number generator. The time at which the request is scheduled followed a Poisson distribution. The requested start time was set to be the time at which the request is scheduled plus a randomly generated lag. Since the results are relatively insensitive to the lag time, we arbitrarily set the mean lag time to be 100 units.

The number of requests in the study interval was set to one of the values 200, 400, 600, 800, and 1000 for the random networks and to one of 100, 200, 300, 400, and 500 for the remaining networks. The allowable mean request durations were 200, 400, 600, 800, and 1000 time units. The allowable mean request bandwidths were 500, 1000, 1500, 2000, and 2500 Mbps for the random networks and 10, 30, 50, 70 and 90 Mbps for the remaining networks. For each setting of the 3 control parameters, we ran 10 trials. In the case of random networks, the network topology was randomly regenerated for each of the 10 trials. For each trial, we measured both RAR and BAR. In reporting our results, we computed the average RAR for all conducted experiments where one of the 3 parameters is fixed.

Our experimental results show that the dynamic alternative feasible path algorithm (DAFP) and the minimum hop feasible path algorithm (MHFP) consistently outperform the other algorithms. For small networks such as the MCI, Cluster, and Burchard networks the dynamic alternative feasible path algorithm (DAFP) gives better performance than MHFP, while on larger networks such as the Abilene network(66 nodes), and the random networks that have 200+ nodes, the two algorithms are comparable. Further, the improvements of the better of MHFP or DAFP over OSPF for a given network ranges from 10% to 15% across a range of requests.

Figures 2 and 3 give the average acceptance ratios for the fixed-slot algorithms as a function of the number of requests in the study interval for a small and a large network, respectively. Generally, the fixed-slot algorithms OSPF, kDP, and kSP that do not guarantee to make a reservation when such a reservation is possible fared worse than the algorithms that provide such a guarantee. However, at times, the performance of the best “no guarantee algorithm” was quite close to or slightly better than that of the worst “guarantee algorithm.” On our non-random networks, OSPF consistently had the worst performance. However, on our random networks, kDP was consistently worst and, often, by quite a margin. As expected, as the network gets saturated (i.e., the number of requests in the study interval increases), the RAR for all algorithms declines and the rate of decline is about the same for all algorithms.

We also found that the relative performance of the algorithms is the same as for the case when we fixed either the
posed in this paper are better than other algorithms (in the sense of maximizing network utilization) on test networks. From the standpoint of algorithmic complexity, MHFP is considerably faster than DAFP, and should be preferred for large networks.

References