 Research Article

JOURNAL OF
Optical Communications
and Networking

Symmetry-free algorithm for spectrum allocation: parallel implementations and evaluation

George N. Rouskas,* Shubham Gupta, and Priya Sharma

North Carolina State University, Raleigh, North Carolina 27606, USA
**rouskas@ncsu.edu*

Received 10 February 2023; revised 5 June 2023; accepted 20 June 2023; published 11 September 2023

Our symmetry-free model for spectrum allocation (SA) in networks of general topology leverages two properties: (1) SA is equivalent to a connection permutation problem, and (2) in assigning spectrum, it is sufficient to consider the allocation made by the first-fit (FF) algorithm. This model opens up algorithmic approaches that altogether sidestep spectrum symmetry, i.e., eliminate from consideration the exponential number of equivalent solutions resulting from spectrum slot permutations. Recursive FF (RFF) is such an algorithm; it applies FF recursively to search the connection permutation space and solve the SA problem optimally. Moreover, parallelism is inherent in the spectrum symmetry-free model, as the connection permutation space may be naturally decomposed into non-overlapping subsets that can be searched independently. Accordingly, RFF admits multithreaded implementations that may be tailored to the computing environment at hand. In this work, we present two strategies for parallelizing the execution of RFF, and we evaluate them experimentally using a comprehensive set of metrics. Our experiments indicate that RFF explores a vast number of symmetry-free solutions, and for moderate-sized networks, it takes mere seconds to yield solutions that are either optimal or very close to the lower bound. © 2023 Optica Publishing Group

https://doi.org/10.1364/JOCN.487181

1. INTRODUCTION

Spectrum allocation (SA) is integral to optical network design and has been studied extensively for decades, usually coupled to other objectives including routing [1–3], traffic grooming [4], network survivability [5], and virtual topology design [6,7]. In the context of elastic optical networks (EONs), the routing and SA (RSA) [8] and the routing, modulation, and SA (RMSA) [9] problems have received considerable attention. Both problems are NP-hard, and a suite of integer linear programming (ILP) formulations have been developed over the years to tackle a wide range of problem variants [2,10–12].

The SA problem is known to be intractable even when considered in isolation, i.e., separately from other aspects of network design [13]. Yet spectrum symmetry, an aspect unique to SA, makes the problem especially challenging to optimal algorithms, including ILP solvers. Spectrum symmetry arises from the fact that slices of optical spectrum are *interchangeable* [14]. Hence, by simply permuting the spectrum slots, every solution (optimal or not) to the SA problem yields an exponential number of equivalent solutions [15]. Conventional ILP formulations cannot account for spectrum symmetry and necessarily encompass this vast number of distinct yet equivalent solutions. Consequently, ILP formulations do not scale [15], and heuristic algorithms have to be employed for networks encountered in practice [16].

A few years ago we developed, an ILP formulation based on maximal independent sets (MISs) [17] for the offline routing and wavelength allocation (RWA) problem in ring networks. The formulation does not suffer from the symmetry problem and obtains optimal solutions to RWA problem instances on maximum size (16-node) SONET rings in seconds. Unfortunately, the number of variables in MIS-based formulations increases exponentially with the network size, and they cannot be applied practically to mesh networks. Thus, a general optimal solution approach to the RWA and RSA problems that avoids spectrum symmetry has so far eluded the research community.

Our recent results regarding the SA problem in isolation (i.e., without the routing dimension) represent a first step towards a general approach to optical network design that sidesteps spectrum symmetry. Our work considered the first-fit (FF) algorithm, a well-known and simple heuristic for the SA problem that has been shown to be effective across a wide range of network topologies and traffic demands [16,18,19]. Our main result was to prove an optimality property of the FF heuristic [20] that allows for the design of symmetry-free algorithms. Specifically, we showed that there exists a permutation of the traffic connections such that applying the FF heuristic to the connections in the order implied by this permutation yields an optimal solution to the SA problem. This optimality property implies that to find an optimal solution, it is sufficient

to consider only the connection permutations, and it is not necessary to account for any allocation of spectrum slots other the one determined by the FF heuristic.

Following up on this insight, we also developed recursive FF (RFF), an optimal branch-and-bound algorithm [20]. RFF searches the space of connection permutations and applies the FF heuristic as it incrementally builds each permutation during the search. While the connection permutation space is itself exponential in size, RFF represents a significant improvement over existing approaches as it completely eliminates spectrum symmetry.

The connection permutation (solution) space can be naturally partitioned into non-overlapping regions that may be explored independently and in parallel. Therefore, in this paper we present two strategies for parallelizing the execution of the RFF algorithm. Given a limit on the amount of time that RFF is allowed to run, the two strategies represent a trade-off between (1) the coverage of the solution space, i.e., the number of distinct regions of the space explored, and (2) the amount of time that the algorithm spends exploring a particular region. We also evaluate the two strategies experimentally using a comprehensive set of metrics that provides insight into the algorithm's operation.

Following the introduction, we define the SA problem we consider in this work and explain spectrum symmetry in Section 2, we discuss the FF spectrum-free property in Section 3, and we review the RFF algorithm in Section 4. We introduce two parallel implementations of RFF in Section 5, and we discuss our experimental setup and evaluation results in Section 6. We conclude the paper in Section 7.

2. SA PROBLEM AND SPECTRUM SYMMETRY

We consider a network with topology graph $G = (V, A)$, where V is the set of nodes, and A is the set of directed fiber links in the network. Let $N = |V|$ denote the number of nodes and $L = |A|$ the number of directed links. Each link supports F spectrum slots, numbered $1, 2, \ldots, F$. The traffic offered to the network consists of a set $\mathcal{C} = \{C_i\}$ of K connection requests. Each connection is a tuple $C_i = (s_i, d_i, p_i, f_i)$, where s_i is the source and d_i the destination node of the connection; p_i is the path between nodes s_i and d_i that the connection must follow; and f_i is the number of spectrum slots required to carry the traffic from s_i to d_i.

In this work, we study the offline SA problem shown in Fig. 1. Specifically, the objective is to allocate spectrum slots to connections so as to minimize the index of the highest slot used on any link, while satisfying the three spectrum constraints listed in Fig. 1. This objective seeks to pack the spectrum slots assigned to the carried traffic as tightly as possible, and hence, it attempts to indirectly minimize spectrum fragmentation and allow for growth in demand; consequently, it is one that has been adopted widely in the literature. Also, we assume that the path p_i of each connection C_i is fixed and pre-determined, i.e., any routing decision has been made before the allocation of spectrum. Therefore, any algorithm that solves this SA problem, including the one we present in this paper, may be applied as part of a multi-step, iterative approach to the RSA problem [16].

The Offline SA Problem
Input:
- A graph $G = (V, A)$
- A set $\mathcal{C} = \{C_i = (s_i, d_i, p_i, f_i), i = 1, \cdots, K\}$, of connection requests

Output: An allocation of f_i spectrum slots to each connection C_i along the physical path p_i
Spectrum Constraints:
- *Contiguity:* each connection C_i is allocated a block of f_i contiguous spectrum slots
- *Continuity:* each connection is allocated the same block of spectrum slots along all links of its path p_i
- *Non-overlap:* connections whose paths share a link are allocated non-overlapping blocks of spectrum slots

Objective: Minimize the index of the highest spectrum slot used on any link in the network

Fig. 1. Offline SA problem.

We have shown [13] that the SA problem is NP-hard even for chain networks with four or more links. But in addition to its theoretical computational intractability, a major practical challenge in tackling the SA problem of Fig. 1, or any of its variants that have been studied in the literature, relates to *spectrum symmetry*. Spectrum symmetry refers to the fact that blocks of contiguous spectrum slots of a certain size are interchangeable. Therefore, for any optimal solution to the SA problem, one can derive a large number of equivalent solutions simply by permuting the spectrum blocks.

Figure 2 illustrates how spectrum symmetry leads to multiple equivalent solutions. Figure 2(a) shows a solution to the SA problem on a four-link chain network with $K = 9$ connections. Each connection is represented by a different color along the links of its path. For instance, the bottommost (light blue) connection spans all four links of the network, indicating that the connection has been allocated this contiguous block of two spectrum slots along each of these links. Note that the solution shown in Fig. 2(a) is optimal: the highest assigned slot on link 3 is equal to the lower bound for this problem instance, i.e., the number of slots required to carry the connections whose path includes link 3.

To illustrate how spectrum symmetry affects the SA problem, consider two blocks of spectrum slots in Fig. 2(a): the three-slot block consisting of spectrum slots 3–5, and the five-slot block consisting of spectrum slots 6–10. Figure 2(b) shows the equivalent solution that we obtain by permuting these two blocks of slots. In the new solution, the two connections that were allocated slots in the range of 3–5 in Fig. 2(a) are now shifted up and are allocated the corresponding slots in the range of 8–10, while the four connections that were allocated slots in the range of 6–10 are now shifted down accordingly. Otherwise, the two solutions in Figs. 2(a) and 2(b) are identical; they are also equivalent in that they yield the same objective function value. Furthermore, note that (1) it is possible to obtain many more solutions equivalent to the two shown in Fig. 2 by permuting different blocks of spectrum slots, and (2) spectrum symmetry also applies to non-optimal solutions.

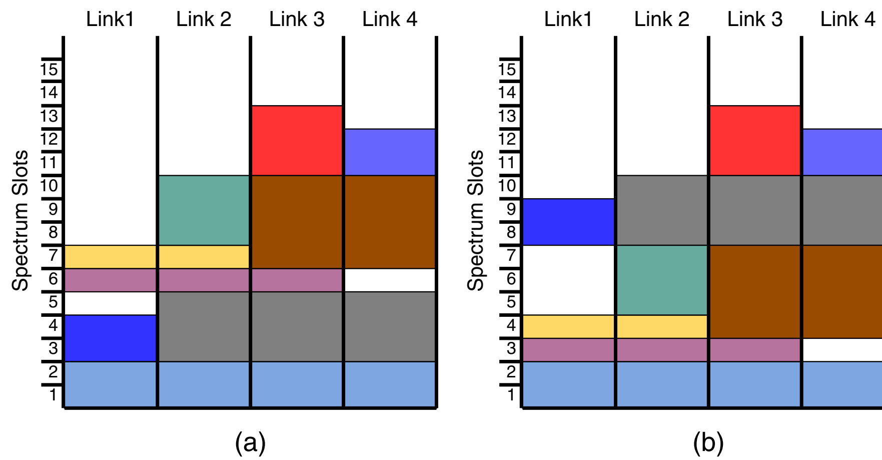

Fig. 2. (a) Optimal solution to an SA problem instance constructed by the FF algorithm. (b) Equivalent solution due to spectrum symmetry; this solution is derived by the FF solution in (a) by permuting certain spectrum slots as explained in the text.

Based on the above discussion it is clear that, due to spectrum symmetry, conventional ILP formulations may yield an exponentially large number of equivalent (optimal or suboptimal) solutions. Consequently, ILP solvers are forced to explore a solution space whose size is essentially the product of the request permutation space and the spectrum permutation space. However, as we explain next, the SA problem can be recast in a way that eliminates equivalent solutions due to symmetry and hence significantly reduces the size of the solution space that needs to be explored.

3. FIRST-FIT SYMMETRY-FREE PROPERTY

Let us return to the offline SA problem shown in Fig. 1 on graph G and request set $\mathcal{C} = \{C_i, i = 1, \ldots, K\}$. Let P be a permutation (i.e., an ordering) of the traffic connections C_i. Let $\mathrm{SOL}(P)$ denote the solution to the SA problem obtained by the FF algorithm when it considers each connection in the order implied by permutation P. Let OPT denote the objective value of an optimal solution to the SA problem. Clearly, for any permutation P of the connections, it must be that $\mathrm{OPT} \leq \mathrm{SOL}(P)$.

We have shown that there exists a permutation $P^{\star}_{\mathrm{FF}}$ of the traffic requests such that applying the FF algorithm to the connections in the order in which they appear in $P^{\star}_{\mathrm{FF}}$ yields an optimal solution to the SA problem [20], i.e., $\mathrm{SOL}(P^{\star}_{\mathrm{FF}}) = \mathrm{OPT}$. We refer to this result as the *optimality property of the FF algorithm.* For instance, it is not difficult to see that the solution shown in Fig. 2(a) is the product of the FF algorithm on an appropriate permutation (ordering) of the $K = 9$ connections. However, the equivalent solution in Fig. 2(b) would not be produced by the FF algorithm; rather, FF would have allocated slots 5 and 6 to the (dark blue) connection spanning just link 1, not slots 8 and 9. This FF optimality property helps explain why many studies of the SA (and WA) problem have confirmed that the FF heuristic yields good solutions across diverse problem instances.

The proof of the FF optimality property, which we presented in [20], is by construction. Specifically, starting with an optimal solution of value equal to OPT, our proof shows how to construct a permutation $P^{\star}_{\mathrm{FF}}$ such that $\mathrm{SOL}(P^{\star}_{\mathrm{FF}}) = \mathrm{OPT}$. We now observe that the construction process of the proof in [20] may be applied to *any* solution to the SA problem, not just an optimal solution. Therefore, we have the following more general result.

Lemma 3.1 (FF Symmetry-Free Property). *Consider any solution S to an instance of the SA problem that achieves an objective value $V_S \geq OPT$, where OPT is the objective value of an optimal solution to this SA instance. There exists a permutation P_{FF} of the connections such that applying the FF algorithm to the connections in the order implied by P_{FF} yields a solution to the SA instance with objective value $SOL(P_{FF}) = V_S$.*

Proof. Applying the construction process of the proof of ([20], Lemma 3.1) to the given solution S will produce the required permutation P_{FF}. ■

Essentially, *the FF symmetry-free property reduces the SA problem to a permutation problem* where the objective is to find the (unknown) permutation $P^{\star}_{\mathrm{FF}}$. Since there is no need to consider an SA for any of the various connection permutations other than the one produced by the FF algorithm, any permutation for which the FF algorithm yields the smallest objective value is an optimal solution to the SA problem instance at hand. We note, however, that in a network with N nodes and traffic between all node pairs, the size K of connection set $\mathcal{C}$ is $O(N^2)$. Therefore, any algorithm that considers all possible permutations of connections to determine the optimal SA must take time that is exponential in the size of the network, $O(N^2!)$.

Since the number of connection permutations is exponential, the *theoretical* complexity of the SA problem is not affected by its transformation into a permutation problem. Nevertheless, the FF symmetry-free property allows us to design algorithms that ignore the exponential number of symmetric solutions derived from spectrum permutations, e.g., solutions such as the one in Fig. 2(b). Doing so drastically reduces the size of the solution space that needs to be explored *in practice*. We discuss such an algorithm, RFF, in the next section.

4. SYMMETRY-FREE RFF ALGORITHM

The RFF algorithm is described in detail in [20]. For the sake of completeness, in this section, we first explain the basic operation of RFF; we introduce two approaches to executing the algorithm in parallel in the following section.

Due to the FF optimality and symmetry-free properties we discussed earlier, an optimal solution to the SA problem can be obtained by applying the FF algorithm to all permutations of the K connections and selecting the best one. RFF [20] is a recursive branch-and-bound algorithm that searches the space of $K!$ connection permutations efficiently. RFF starts with an initial permutation, P, that is provided as input. The algorithm applies the FF heuristic on P to obtain an initial solution to the SA problem, and records this solution as the best one it has found so far. Then, RFF calls itself recursively and modifies the initial permutation to create additional ones. Each call incrementally builds a new permutation one connection at a time. A recursive call also applies the FF algorithm to the partial permutation built so far. Once RFF has built a complete permutation (i.e., one consisting of some ordering of all K connections), if the solution of this permutation is better than the best one it has found so far, it records it as the new best solution. If at any point during the recursion RFF determines that the solution to the current partial permutation is not better than the best known solution, it abandons further exploration along the current path and backtracks.

Figure 3 shows the tree of recursive calls that RFF makes on an instance with four connections, starting with the initial permutation $P = [A, B, C, D]$, shown as "tentative" in the figure. Starting with the initial call at the root of the tree, RFF proceeds in a depth-first manner along the leftmost path of the tree. The call representing the leftmost child of the root fixes (finalizes) the first connection (A) in the permutation, and proceeds recursively to finalize the remaining three connections. The leaves of the tree represent the distinct permutations, and hence, there are $K!$ leaves. In the example of Fig. 3, the leftmost subtree of the root has $3! = 6$ leaves corresponding to the six permutations in which connection A is in the first position (as in the root of this subtree). The other three subtrees also have six leaves (not shown in the figure), for a total of $4! = 24$ permutations.

We say that RFF *directly* explores a permutation if it reaches the leaf of the tree representing this permutation. In this case, RFF computes the FF solution on the permutation and compares it to the current best solution to determine whether it is better. However, branch-and-bound algorithms such as RFF do not need to visit all leaves and explore directly all possible solutions. As we mentioned earlier, while traversing a path to a leaf, RFF may determine that the partial permutation it has constructed cannot improve on the current best solution. Then, RFF abandons further direct exploration of the current subtree and backtracks to start on a different permutation. In this case, we say that RFF has *indirectly* explored all the leaves (permutations) of the abandoned subtree, as it has made a determination that they do not represent better solutions than the one it has recorded. The number of permutations (i.e., leaves of the subtree) indirectly explored at the time RFF backtracks can be calculated by keeping track of the height of the node where the backtracking occurred.

Overall, the RFF algorithm represents both a substantial departure from, and a meaningful improvement over, existing methods and algorithms in searching for an optimal solution to the SA problem because of several new and unique features:

- it is spectrum symmetry free, i.e., it does not consider any equivalent solutions that are simply the result of spectrum slot permutations;
- it applies the simple, well-understood, and widely adopted FF algorithm;
- it uses indirect exploration to eliminate whole subtrees of the solution space without explicitly visiting all the permutations they contain;
- it makes it possible to calculate precisely the size of the solution (permutation) space it has explored at any point in time; and
- it can be executed in parallel.

Next, we discuss in detail parallel implementation aspects of the RFF algorithm.

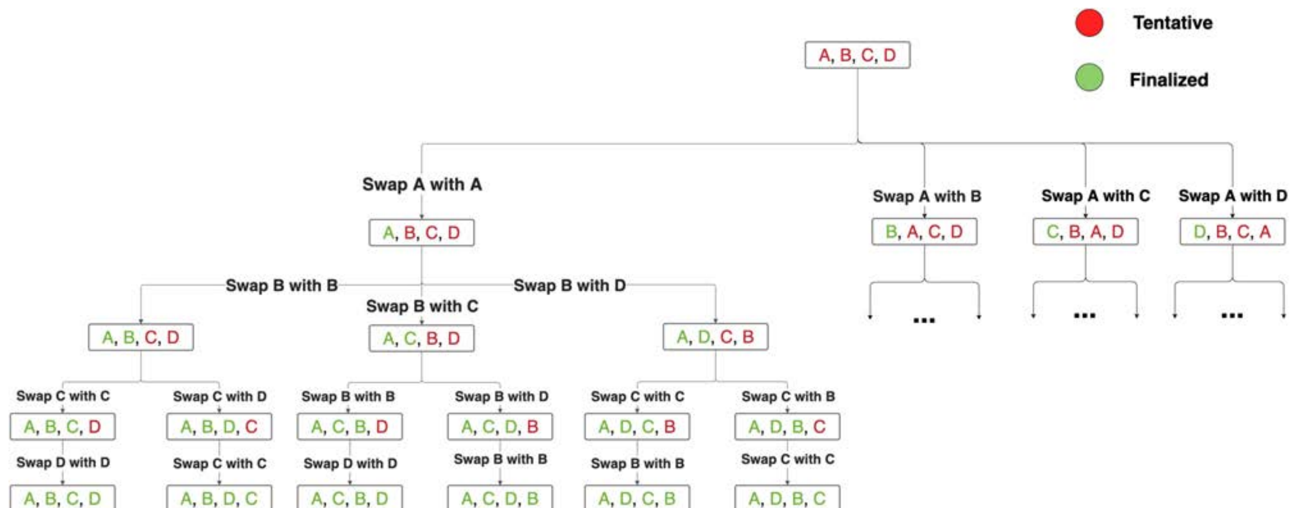

Fig. 3. Tree of RFF calls on a set of four connections. The root of the tree represents the initial call with initial permutation $P = [A, B, C, D]$.

5. PARALLEL IMPLEMENTATIONS OF RFF

Observe from Fig. 3 that a recursive call starting at some node X of the RFF tree simply proceeds along a path from X to a leaf node by modifying the trailing "tentative" portion of the permutation it is given as input. As a result, any pairs of recursive calls that operate in non-overlapping subtrees do not interact with each other, with one exception: calls that reach a leaf node check, and possibly update, the value of the current best solution. Therefore, RFF may be parallelized by (1) locking access to the variable holding the best known solution and (2) assigning recursive calls in non-overlapping subtrees to different threads. These threads will execute in parallel and will interfere with each other only when they need to access or update the locked variable. Importantly, when one thread finds a better solution and updates the variable, it allows all threads to eliminate subtrees of solutions earlier; in turn, this property leads to faster than linear indirect exploration of the permutation space as the number of threads increases.

Consider an instance of the SA problem with K requests to be solved using RFF, and assume that we may deploy at most M threads in parallel. The limit on the number of threads may be imposed by the operating system or the computational budget (i.e., the amount of computational resources available for tackling the problem). We also assume that there is an upper bound T on the amount of time that the algorithm is allowed to run. Note that the number K of requests corresponds to the number of node pairs in the network. Since K is $O(N^2)$, where N is the number of network nodes, we assume that $M < K$ for network topologies encountered in practice. Therefore, in designing parallel implementations for the RFF algorithm, the objective should be to: *determine how to deploy the M threads so that the algorithm will return a solution of good quality within T time units*.

To this end, a parallelization strategy must ensure that

(1) the algorithm explores the largest possible subset of the solution (permutation) space, and
(2) the subset explored is representative of the entire solution space and is not limited to the neighborhood around the initial permutation provided as input.

To gain insight into how to achieve the second goal above, recall that a recursive call of RFF that starts at the root of a subtree (refer to Fig. 3) will initially follow the leftmost path to a leaf. After reaching a leaf or abandoning the current path because it will not lead to a better solution, the algorithm backtracks and follows the downward path immediately to the right of the one it was previously on. Therefore, RFF visits the leaves (or subtrees) of the current subtree in the order from left to right. Consequently, if the time allotted is not sufficient for RFF to visit the whole subtree, it will visit its leftmost subtrees and leave the rightmost subtrees unexplored; the relative fraction of visited and unexplored subtrees depends on the time the algorithm is allowed to run. Furthermore, our initial findings in [20], which we will confirm in the following section, indicate that when exploring a particular part of the solution space, RFF in general finds good solutions quickly and spends most of the remaining time exploring subtrees that do not improve on the early solutions.

As a second observation from Fig. 3, we note that the root of the RFF tree has K subtrees, each subtree exploring the permutation sub-space in which a specific connection is fixed in the first position. Therefore, given that $M < K$ (and often that $M \ll K$ as K is $O(N^2)$), it is not possible to explore all K subtrees in parallel simultaneously with just M threads. More generally, it is difficult to divide the permutation space in M roughly equal and non-overlapping subsets that can be explored independently and in parallel.

Based on these observations, our approach is to deploy multiple batches of M threads, each batch running for an amount of time $t < T$, such that (1) collectively all the batches cover many diverse areas of the solution space, and (2) the total running time is equal to T. Specifically, we consider two strategies to parallelizing RFF that differ in the depth of the RFF recursion call tree, shown in Fig. 3, at which threads are deployed to execute in parallel. The two strategies represent a trade-off between (1) the amount of time the algorithm spends exploring a particular part of the solution space and (2) the number of distinct (non-overlapping) regions of the solution space that the algorithm visits.

A. Depth-0 Parallelization Strategy

As we mentioned above, the subtrees of the root divide the solution space into K ($K = 4$ in Fig. 3) mutually disjoint and collectively exhaustive subsets of the solution space; hence, this is a natural point to introduce parallelism. Since we assume that $M < K$, we run RFF as follows:

(1) Let $m = \lceil K/M \rceil$ be the number of batches of M parallel threads needed to explore all K subtrees of the root (the mth batch may have fewer than M threads).
(2) Let $t = T/m$ be the running time for each batch of threads.
(3) Start a batch of M threads that run in parallel, each thread starting execution at one of the M leftmost subtrees (children) of the root. Terminate this batch of threads after t time units, and record the best solution found.
(4) Repeat step (3) after passing the best solution to the next batch of threads that start at the next leftmost M subtrees of the root, and so on, until the last batch of threads starting at the rightmost subtrees of the root completes at time T.

This Depth-0 parallelization strategy is a natural and straightforward approach, it is easy to implement in practice, and it will explore all K subtrees of the root that correspond to non-overlapping subsets of the solution space. Nevertheless, a disadvantage of the strategy is that the thread at each subtree of the root will run for a relatively small time t, and for values of K corresponding to typical networks, it will not be able to explore the whole subtree that contains $(K-1)!$ leaves. Specifically, each thread will explore only the leftmost part of its subtree; hence, this strategy will miss the regions of the solution space that lie in the rightmost parts of the subtrees of the root.

B. Depth-1 Parallelization Strategy

Our second strategy aims to explore the entire solution space more uniformly by introducing parallelism at depth 1 of the tree. Under this strategy, each thread explores a subtree of a child of the root, and hence, it starts execution at a grandchild node of the root. Since each thread starts at a lower level of the tree, the subset of solutions explored will be more evenly distributed across the entire space than with the Depth-0 strategy.

The root of the RFF tree has K children, each of which has $K-1$ subtrees, for a total of $K(K-1)$ subtrees at depth 1. Assuming that the available total computational time T and/or the number M of parallel threads are sufficiently large, we can deploy $m = \lceil K(K-1)/M \rceil$ batches of threads, each batch running for $t = T/m$ time units. However, the number of subtrees at depth 1 is $O(N^4)$ and can easily grow to millions or beyond for medium- to large-sized networks; for a 40-node network with traffic between every pair of nodes, the number of subtrees at depth 1 exceeds 2 million. Therefore, unless the total time T can be very long, the running time t for each batch of threads will be too small to meaningfully explore its subtree. In this case, it may be necessary to explore only a subset of the subtrees at depth 1 to remain within the available computational budget. Clearly, there are various approaches for sampling the solution space at depth 1. For instance, the subtrees to be explored may be selected randomly, or they could be selected deterministically in a way that uniformly spreads the subtrees across the solution space. Although an in-depth evaluation of sampling strategies may provide further insight, it is outside the scope of our work. In the next section, we present a deterministic strategy that worked well for the 32-node GEANT2 topology we used in our experiments.

Finally, we note that it is possible to extend the two parallelization strategies so as to introduce parallelism at lower levels of the RFF tree, e.g., a Depth-2 or Depth-3 strategy. Since the number of subtrees to be explored increases by a factor of $O(N^2)$ for each lower level, a sampling strategy becomes even more significant in this case. Exploring the benefits of introducing parallelism below depth 1 of the RFF tree is reserved for future research.

C. Illustrative Example

Let us illustrate the operation of the Depth-0 and Depth-1 strategies by referring to the tree of RFF calls in Fig. 3, assuming that $M = 2$ threads may be executed in parallel. With the Depth-0 strategy, the original call to RFF starts at the root of the tree and immediately spawns two threads: one to explore the leftmost subtree of the root (i.e., the subtree in which A is fixed as the first connection of the permutation), and one to explore the subtree immediately to the right (i.e., the one in which B is fixed as the first connection of the permutation). The two threads proceed to explore their respective subtrees independently and in parallel. A thread terminates when either (1) it backtracks to the root of its subtree (in which case it has completely explored its part of the permutation space), or (2) a time limit for its execution is reached. As soon as a thread terminates, the original call spawns a new thread to explore the next subtree of the root, in this case the one in which C is fixed as the first connection of the permutation. This process continues until all subtrees of the root have been explored, with the original call keeping track of the status of all threads and making sure that M threads are executing in parallel at all times to take maximum advantage of parallelism.

The operation of the Depth-1 strategy is similar except that each thread explores a subtree of a child of the root. Specifically, the original call to RFF starts at the root of the tree, and then proceeds (as in a single-threaded implementation) in a depth-first search (DFS) manner to explore the leftmost child of the root. At that node, the original call spawns $M = 2$ threads: one to explore the leftmost subtree of the node (i.e., the one in which A and B are fixed as the first two connections of the permutation), and one to explore the subtree immediately to the right (i.e., with A and C fixed as the first two connections of the permutation). The two threads proceed to explore their part of the permutation space until they terminate, as explained above. When a thread terminates, the original call spawns another to explore the next subtree to the right, in this case, the one in which A and D are fixed as the first two connections of the permutation. As threads complete, the original call backtracks, visits the following children of the root, and spawns threads to explore their subtrees. In this example, when the second thread completes, the original call will backtrack to the second child of the root from the left and assign a new thread to the subtree (not shown in Fig. 3) with B and A fixed as the two first connections of the permutation.

6. NUMERICAL RESULTS

A. Simulation Setup

We have performed simulation experiments to evaluate the performance of the RFF algorithm under the two parallel execution strategies we discussed in the previous section. We run the experiments on the Henry2 Linux HPC cluster at NC State University [21–25], which consists of more than 1000 compute nodes and over 10,000 cores.

The experimental setup is similar to the one we used in [20]. Specifically, we consider two network topologies, the 14-node, 21-link NSFNET and the 32-node, 54-link GEANT2 shown in Fig. 4, with all link lengths equal to 1. Each connection is carried along the shortest (i.e., minimum hop) path between its source and destination nodes. We create SA problem instances by generating traffic demands between all node pairs in each network. Specifically, we assume that data rates may take values (in Gbps) from the set {10, 40, 100, 400, 1000}, and for each problem instance, we generate a random value for the demand between a pair of nodes based on one of three distributions:

- *uniform*: each of the five rates is selected with equal probability;
- *skewed low*: the rates above are selected with probability 0.30, 0.25, 0.20, 0.15, and 0.10, respectively; or
- *skewed high*: the five rates are selected with probability 0.10, 0.15, 0.20, 0.25, and 0.30, respectively.

Given the traffic rates between each node pair, we calculate the corresponding spectrum slots by assuming that the slot width is 12.5 GHz, and adopting the parameters of [25] to determine the number of spectrum slots that each connection

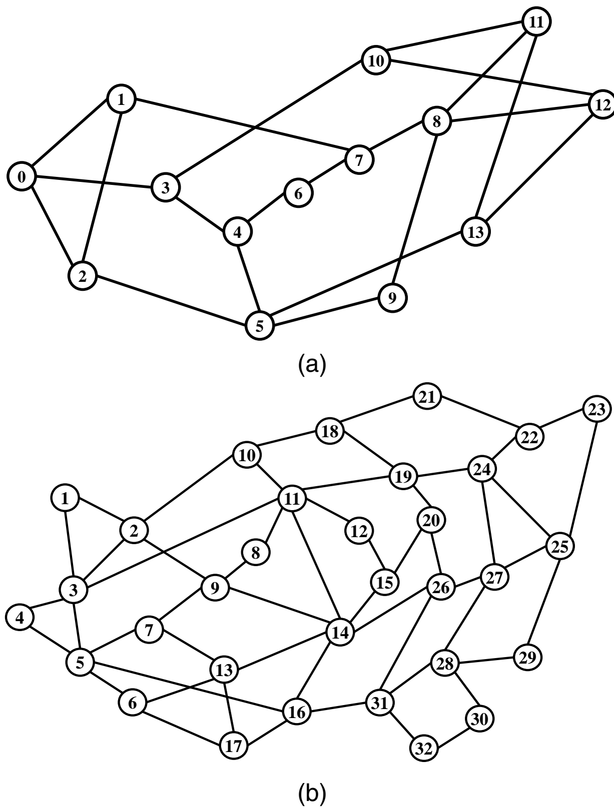

Fig. 4. Network topologies used in our study: (a) NSFNET, (b) GEANT2; all link lengths are equal to 1.

requires based on its data rate and path length. For each traffic distribution, we generate 100 random problem instances.

We consider the highest index of allocated spectrum slots on any network link as the performance metric of interest. To allow for a meaningful comparison between different problem instances, we *normalize* the solutions with respect to the lower bound; in other words, we divide the absolute value returned by the algorithm with the lower bound for the corresponding instance. We obtain a lower bound LB on the optimal objective value by ignoring the spectrum contiguity and continuity constraints and simply counting the spectrum slots required by all connections on the most congested link.

In all experiments, we deployed $M = 32$ parallel threads, the maximum number available to us on the Henry2 cluster. NSFNET problem instances have $K = 91$ connections; hence, the root of the RFF tree has 91 children (subtrees) and $91 \times 90 = 8190$ grandchildren. We set $T = 270$ min as the bound on the running time. For the Depth-0 strategy, we deployed $m = \lceil 91/32 \rceil = 3$ batches of parallel threads, each batch running for $t = 90$ min. For the Depth-1 strategy, we used $m = \lceil 8190/32 \rceil = 256$ batches of parallel threads; we let each batch run for $t = 1$ min for a total of 256 min of running time. With this arrangement, RFF runs for approximately the same amount of time under both strategies.

In the case of GEANT2, we have $K = 496$. The Depth-0 strategy requires $m = \lceil 496/32 \rceil = 16$ batches of 32 threads; therefore, we set $T = 80$ min as the bound on running time and let each batch run for $t = 5$ min. [Although we set the running time bound for the GEANT2 topology at a lower value (80 min) than for the NSFNET network (270 min), using a longer running time is unlikely to have produced better results for GEANT2. Specifically, as Fig. 10 shows below, RFF typically finds the best solution within seconds, and in the worst case, in 42 min, i.e., well before the time bound is reached. Moreover, the GEANT2 solutions found are very close to the lower bound, and hence, allowing for additional running time would make a marginal difference at best.] The Depth-1 strategy, on the other hand, would need to explore $495 \times 496 = 245{,}520$ subtrees and, hence, it would require $m = 7673$ batches of 32 threads; each batch would then need to run for less than 1 s for the total running time to be no more than 80 min. Instead, we decided to modify the strategy as follows. Rather than considering all children nodes of the root, we consider only 80 children uniformly spaced apart, i.e., children 1, 7, 13, …, 475. For each of these children, we create a batch of 32 threads, each thread exploring a subtree of the child node. Therefore, we deployed only $m = 80$ batches of parallel threads in this case and explored only $80 \times 32 = 2560$ grandchildren of the root instead of 245,520. We let each batch run for $t = 1$ min so that the total running time T is the same for both strategies. Although this approach does not explore the whole space at depth 1 of the tree, it allows us to compare the two strategies without having to use a very long running time.

Table 1 lists the values for the various parameters we used in the simulations.

Table 1. Values of the Parameters Used in the Simulations[a]

			Depth-0		Depth-1	
Topology	M	T	m	t	m	t
NSFNET	32	270 min	3	90 min	256	1 min
GEANT2	32	80 min	16	5 min	80	1 min

[a] M, number of parallel threads; m: number of BATCHES of parallel threads; T, total running time; t, running time for each batch.

B. Results and Discussion

We evaluate the RFF algorithm using several performance measures: (1) the solution quality relative to the FF algorithm and the lower bound LB, (2) the number of permutations (i.e., amount of solution space) it explores, and (3) the amount of time it takes to reach the best solution.

1. Solution Quality

Figures 5 and 6 summarize the results we have obtained for the NSFNET and GEANT2 topologies, respectively. The figures show how far the solutions obtained by the FF, RFF Depth-0, and RFF Depth-1 algorithms are from the lower bound. Each value in the figures represents an average over 100 problem instances for the stated network, algorithm, and traffic distribution; the plots also include 95% confidence intervals. We first observe that, on average, the FF algorithm produces solutions that are within 9%–12% (respectively, 3%–7%) of the lower bound for the NSFNET (respectively, GEANT2) topology. This is consistent with earlier research indicating that FF finds solutions of good quality.

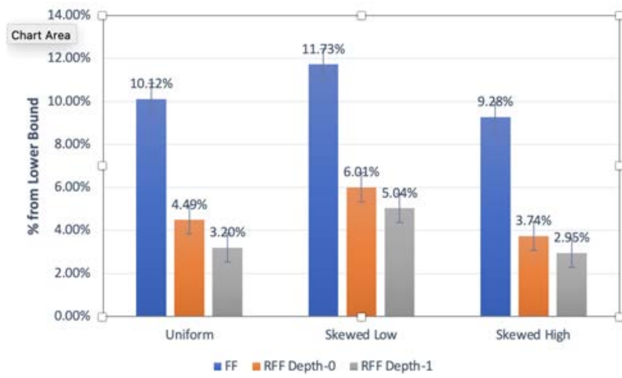

Fig. 5. FF and RFF solution quality as % from LB, NSFNET.

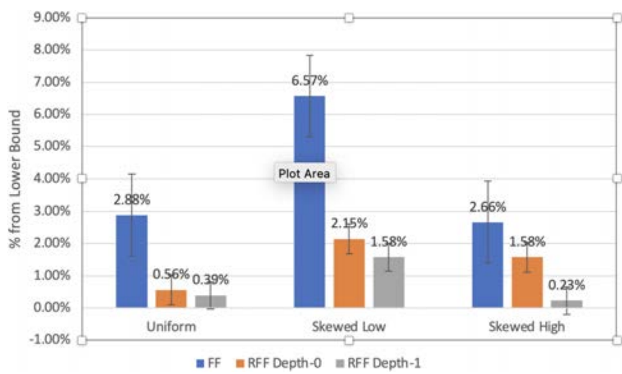

Fig. 6. FF and RFF solution quality as % from LB, GEANT2.

Turning our attention to the RFF algorithm, we notice that the Depth-0 strategy yields solutions that are noticeably better, on average, than the FF solutions across all network topologies and traffic distributions we used in our experiments. Again, this result is in agreement with our findings in [20], despite the fact that those results were obtained using a different implementation of the algorithm than the one we used here. The improvement in solution quality is particularly impressive for the GEANT2 network, as it was achieved even though the FF solutions are very close to the lower bound.

Finally, we see that, on average, the Depth-1 strategy yields a further improvement on solution quality, for both networks and the three traffic distributions. In all our experiments, we have observed that the Depth-1 strategy consistently outperforms the Depth-0 strategy; we discuss this result in more detail at the end of the section.

Figures 7 and 8 present a different perspective regarding the quality of the solutions obtained for the NSFNET and GEANT2 topologies, respectively. Specifically, the two figures plot the number of problem instances for which the Depth-0 and Depth-1 solutions are either (1) better than the corresponding FF solution (denoted as "<FF" in the figures) or (2) equal to the lower bound LB of the corresponding instance (denoted as "= LB"). To interpret the results in the two figures, we note that a solution equal to LB is an optimal one; furthermore, the RFF algorithm starts with the FF solution, and hence, for any instances that it cannot find a better solution, it returns the FF solution.

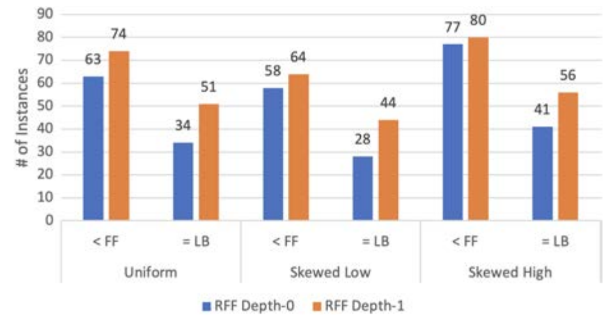

Fig. 7. RFF solutions relative to FF and LB, NSFNET.

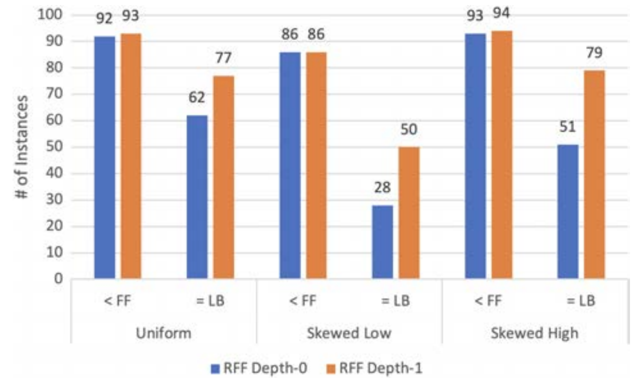

Fig. 8. RFF solutions relative to FF and LB, GEANT2.

Overall, we observe that RFF with the Depth-0 strategy improves upon the FF solution in 58%–77% of the problem instances in the case of the NSFNET, and in 86%–93% of the instances in the case of GEANT2, depending on the traffic distribution. Furthermore, it finds an optimal solution in 28%–41% (respectively, 28%–62%) of the instances for NSFNET (respectively, GEANT2), again depending on the traffic distribution. The Depth-1 strategy achieves better performance, and both the number of instances for which an optimal solution is found and those with a solution better than that of FF are higher than with the Depth-0 strategy. There is only one exception: for the GEANT2 topology with the skewed low distribution, the number of instances with a solution better than that of FF is the same (86) under both strategies; but even in this case, 22 of these instances show an improvement as the Depth-1 strategy is able to find an optimal solution for them.

As a final observation, the results in Figs. 6 and 8 indicate that, for the GEANT2 problem instances we considered in this study, the Depth-1 strategy is effective in finding solutions that are very close to the lower bound (and hence, the optimal value). Therefore, we expect that introducing parallelism at a lower depth of the RFF tree (e.g., by deploying a Depth-2 strategy) would not have significant benefits. Instead, if additional computational time were available (i.e., beyond the $T = 80$ min we used in our experiments), a better option would be to deploy additional batches of parallel threads so as to explore more subtrees at depth 1. For the NSFNET topology, on the other hand, the results in Figs. 5 and 7 show a larger gap between the solutions of the Depth-1 strategy

and the lower bound. Recall also that due to the smaller number K of connections, the Depth-1 strategy for NSFNET explores all subtrees at depth 1. Combined with our later observations regarding the time to find the best solution, these results suggest that, say, a Depth-2 parallelization strategy that explores more regions of the permutation space might be able to find better solutions. Investigating the benefits of a Depth-2 strategy will be the subject of future research.

2. Coverage of the Solution Space

One of the unique features of the RFF algorithm is that it is possible to calculate precisely the number of permutations it explores. Recall from our earlier discussion that RFF explores (evaluates) a permutation in one of two ways: either directly, when it visits the leaf of the tree representing the permutation, or indirectly, when it backtracks and abandons further exploration of the subtree where the leaf representing this permutation resides. In the latter case, all leaves of the abandoned subtree represent solutions that are no better than the one the algorithm has already found. The height of the node at which the algorithm backtracks allows us to calculate the number of leaves in the abandoned subtree, and hence the number of permutations that are indirectly explored at that point.

Tables 2 and 3 list the number of permutations that the RFF algorithm explores in the case of the NSFNET and GEANT2 networks, respectively. To put these figures in perspective, note that the number of permutations is $91! \approx 1.35E140$ for NSFNET and $496! \approx 1.98E1123$ for GEANT2. The values shown are averages that are taken separately over the problem instances for which the algorithm (1) reached the lower bound or (2) did not reach the lower bound.

It is evident from the two tables that the RFF algorithm evaluates an immense number of permutations for either network, the vast majority of which are explored indirectly. Nevertheless, the absolute numbers are impressive given that the algorithm runs for either 270 min (NSFNET) or 80 min (GEANT2). There are also several trends that can be observed from the tables. First, the algorithm explores a larger number of permutations when it cannot reach an optimal solution, as in this case it runs for the entire allotted time T, whereas it terminates as soon as it finds an optimal solution (i.e., one equal to the lower bound), usually well before time T. Second, the algorithm also explores a larger number of solutions under the Depth-1 strategy compared to the Depth-0 strategy. Since the Depth-1 strategy yields better solutions, the algorithm determines earlier (i.e., closer to the root) that a subtree does not contain better solutions, and hence, it eliminates larger subtrees with a larger number of poor solutions. Finally, the algorithm evaluates a number of permutations in the GEANT2 topology that is orders of magnitude greater compared to that for the NSFNET topology, despite the fact that the running time for the former is shorter. Again, this is due to the fact that the subtrees eliminated in the GEANT2 case are far larger than those eliminated in the NSFNET case: the height of the GEANT2 tree is 496, whereas the height of the NSFNET tree is only 91.

Table 2. Number of Permutations Explored, NSFNET

	Reached LB		Did Not Reach LB	
Traffic	**Depth-0**	**Depth-1**	**Depth-0**	**Depth-1**
Uniform	9.9E + 109	7.4E + 114	8.9E + 116	3.0E + 120
Skewed low	3.9E + 124	2.4E + 130	9.9E + 126	2.6E + 127
Skewed high	1.0E + 93	5.7E + 109	2.9E + 108	2.0E + 116

Table 3. Number of Permutations Explored, GEANT2

	Reached LB		Did Not Reach LB	
Traffic	**Depth-0**	**Depth-1**	**Depth-0**	**Depth-1**
Uniform	2.4E + 289	1.9E + 320	2.5E + 363	9.2E + 409
Skewed low	4.2E + 305	9.0E + 325	9.9E + 518	5.7E + 475
Skewed high	7.5E + 286	6.6E + 199	7.9E + 384	2.2E + 327

3. Time to Best Solution

Figures 9 and 10 provide insight into the running time of the algorithm for the two networks. Specifically, the figures plot the minimum, median, average, and maximum time (in seconds) that the algorithm takes to reach the best solution it can find (i.e., the one that it returns upon termination) for each of the two parallelization strategies; all values have been rounded to the nearest integer. For the NSFNET, the algorithm takes a median of 4 s or 5 s, depending on the strategy, to find the best solution. For the Depth-0 strategy, it takes the algorithm at most 77 s to find the best solution across all 300 instances; the rest of the time, the algorithm explores regions of the permutation space that do not contain better solutions. For the Depth-1 strategy that yields lower solutions, it takes about 25 min in the worst case to find the best solution for a handful of outlier instances. In the GEANT2 case, the median is around 42 s for both strategies and all three traffic distributions, and the maximum is around 42 min. In other words, the algorithm finds the best solution rather quickly even for the larger GEANT2 network, and the parallel threads spend most of their time exploring unfavorable permutations.

Recall that under the Depth-1 strategy, each batch of threads runs only for a small amount of time, namely, 1 min, and yet this strategy outperforms the Depth-0 strategy in which each batch runs for a significantly longer amount of time. The results shown in Figs. 9 and 10 offer a likely explanation for this outcome. Specifically, given that the algorithm finds good solutions within seconds regardless of the parallelization strategy, it does not pay off to continue the search within the same area of the solution space for a long time. Each thread in the Depth-0 strategy spends a significant amount of time in the same solution neighborhood, and is trapped at a local minimum. Since the total number of threads at Depth-0 is relatively small (i.e., equal to K), the strategy cannot explore many diverse parts of the solution space. With the Depth-1 strategy, on the other hand, each thread spends a small amount of time in its region of the space, just enough to find a good solution if one exists. Since the number of threads is significantly larger and they start lower in the tree, they cover many more parts of the permutation space and, hence, the strategy is able to find better solutions that the Depth-0 strategy cannot reach. These findings indicate that, whenever either (1) a large degree of parallelization is possible or (2) a large amount of total running

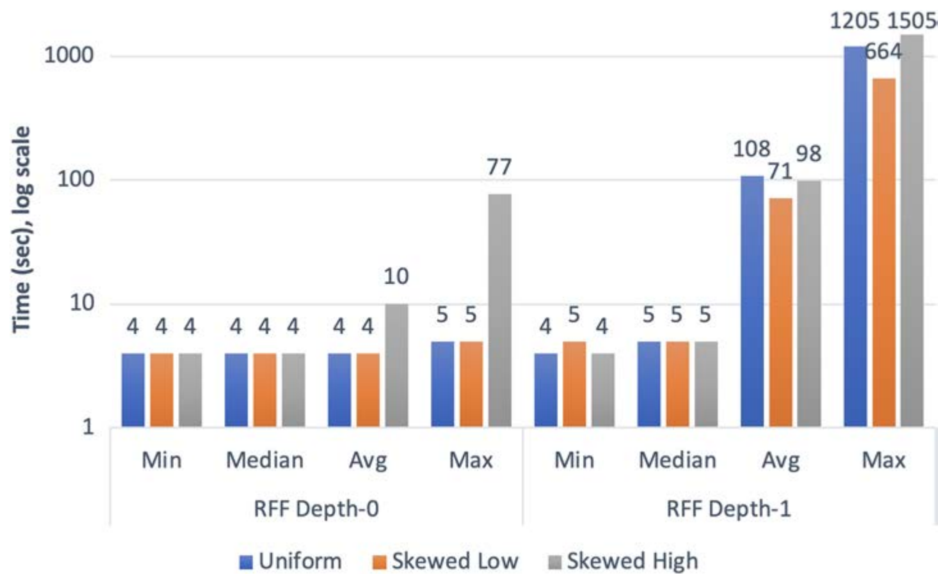

Fig. 9. Time (sec) to reach the best solution, NSFNET.

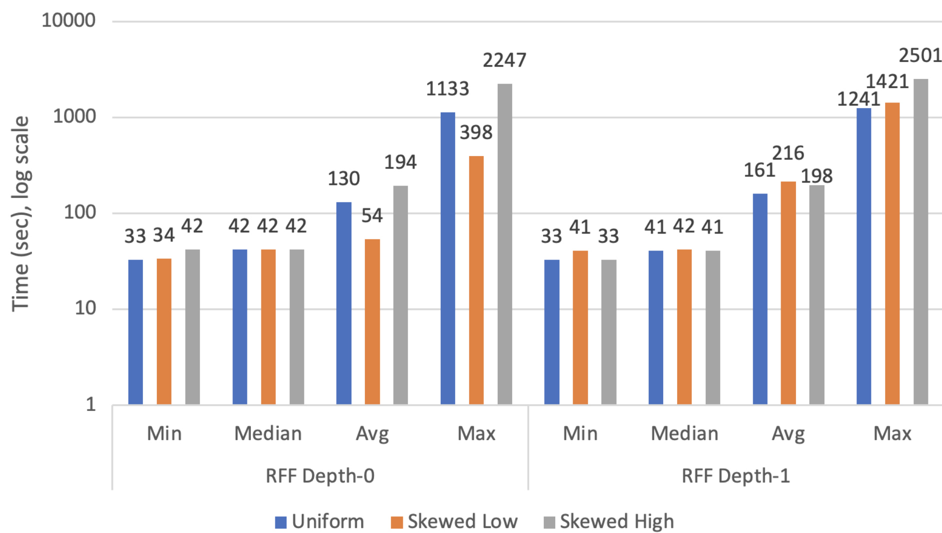

Fig. 10. Time (sec) to reach the best solution, GEANT2.

time T can be afforded, a parallelization strategy that starts at a higher depth and has each thread execute for a small amount of time (1 min or even less) is likely to yield better results.

Overall, our experiments demonstrate that the RFF algorithm explores a vast number of symmetry-free solutions and yields optimal or near-optimal solutions in seconds for networks of moderate size. The algorithm is amenable to parallelization, and since it simply applies the well-known and widely adopted FF algorithm, it can be readily implemented in production environments.

7. CONCLUDING REMARKS

We have presented and evaluated two strategies to parallelize the execution of the RFF, a symmetry-free optimal algorithm for the offline SA problem. Our experiments indicate that parallel implementations of RFF (1) explore vast amounts of the solution space, (2) yield solutions at or near the lower bound, and (3) are able to find good solutions within seconds. Our current research aims to advance this work in two directions. First, we plan to explore the potential of RFF in larger-sized networks by extending the parallelization strategies we developed to depth 2 or lower of the RFF tree. Second, the RFF algorithm is quite different from existing solutions to the SA problem. Therefore, our goal is to use RFF as a starting point to design new approaches for the more general RSA problem; one option is to combine RFF with the routing algorithms we developed in [26,27] so as to tackle large RSA problems efficiently.

Funding. National Science Foundation (CNS-1907142).

Acknowledgment. Portions of this work were presented at IEEE GLOBECOM in 2022 as "Experimental evaluation of a symmetry-free parallel algorithm for spectrum allocation" [28].

REFERENCES

1. B. Jaumard, C. Meyer, and B. Thiongane, "Comparison of ILP formulations for the RWA problem," Opt. Switching Netw. **4**, 157–172 (2007).
2. M. Klinkowski, P. Lechowicz, and K. Walkowiak, "Survey of resource allocation schemes and algorithms in spectrally-spatially flexible optical networking," Opt. Switching Netw. **27**, 58–78 (2018).
3. S. Talebi, F. Alam, I. Katib, M. Khamis, R. Khalifah, and G. N. Rouskas, "Spectrum management techniques for elastic optical networks: a survey," Opt. Switching Netw. **13**, 34–48 (2014).
4. B. Chen, G. N. Rouskas, and R. Dutta, "Clustering methods for hierarchical traffic grooming in large-scale mesh WDM networks," J. Opt. Commun. Netw. **2**, 502–514 (2010).
5. D. Zhou and S. Subramaniam, "Survivability in optical networks," IEEE Netw. **14**, 16–23 (2000).
6. R. Dutta and G. N. Rouskas, "A survey of virtual topology design algorithms for wavelength routed optical networks," Opt. Netw. **1**, 73–89 (2000).
7. H. Wang and G. N. Rouskas, "Hierarchical traffic grooming: a tutorial," Comput. Netw. **69**, 147–156 (2014).
8. K. Christodoulopoulos, I. Tomkos, and E. Varvarigos, "Routing and spectrum allocation in OFDM-based optical networks with elastic bandwidth allocation," in *Proceedings of IEEE GLOBECOM* (2010).
9. F. Shirin Abkenar and A. Ghaffarpour Rahbar, "Study and analysis of routing and spectrum allocation (RSA) and routing, modulation and spectrum allocation (RMSA) algorithms in elastic optical networks (EONs)," Opt. Switching Netw. **23**, 5–39 (2017).
10. F. Bertero, M. Bianchetti, and J. Marenco, "Integer programming models for the routing and spectrum allocation problem," TOP **26**, 465–488 (2018).
11. L. Velasco, M. Ruiz, and M. Klinkowski, *The Routing and Spectrum Allocation Problem* (Wiley, 2017), chap. 3, pp. 43–60.
12. J. Wang, M. Shigeno, and Q. Wu, "ILP models and improved methods for the problem of routing and spectrum allocation," Opt. Switching Netw. **45**, 100675 (2022).
13. S. Talebi, E. Bampis, G. Lucarelli, I. Katib, and G. N. Rouskas, "Spectrum assignment in optical networks: a multiprocessor scheduling perspective," J. Opt. Commun. Netw. **6**, 754–763 (2014).
14. R. Ramaswami and K. Sivarajan, "Routing and wavelength assignment in all-optical networks," IEEE/ACM Trans. Netw. **3**, 489–500 (1995).
15. B. Jaumard, C. Meyer, and B. Thiongane, "ILP formulations for the routing and wavelength assignment problem: symmetric systems," in *Handbook of Optimization in Telecommunications* (Springer US, 2006), pp. 637–677.
16. J. Simmons and G. N. Rouskas, "Routing and wavelength (spectrum) allocation," in *Springer Handbook of Optical Networks*, B. Mukherjee, I. Tomkos, M. Tornatore, P. Winzer, and Y. Zhao, eds. (Springer, 2020).
17. E. Yetginer, Z. Liu, and G. N. Rouskas, "Fast exact ILP decompositions for ring RWA," J. Opt. Commun. Netw. **3**, 577–586 (2011).
18. H. Zang, J. P. Jue, and B. Mukherjee, "A review of routing and wavelength assignment approaches for wavelength-routed optical WDM networks," Opt. Netw. **1**, 47–60 (2000).
19. Y. Zhu, G. N. Rouskas, and H. G. Perros, "A comparison of allocation policies in wavelength routing networks," Photonic Netw. Commun. **2**, 267–293 (2000).
20. G. N. Rouskas and C. Bandikatla, "Recursive first fit: a highly parallel optimal solution to spectrum allocation," J. Opt. Commun. Netw. **14**, 165–176 (2022).
21. NC State, "Henry2 Linux cluster," https://projects.ncsu.edu/hpc/About/ComputeResources.php.
22. L. E. Jackson, G. N. Rouskas, and M. F. M. Stallmann, "The directional p-median problem: definition, complexity, and algorithms," Eur. J. Oper. Res. **179**, 1097–1108 (2007).
23. C. Castillo, G. N. Rouskas, and K. Harfoush, "Efficient resource management using advance reservations for heterogeneous grids," in *Proceeding of the 22nd IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, April 2008.
24. V. Sivaraman and G. N. Rouskas, "HiPeR-ℓ: a high performance reservation protocol with ℓook-ahead for broadcast WDM networks," in *Proceedings of IEEE INFOCOM*, April 1997, pp. 1272–1279.
25. M. Jinno, B. Kozicki, H. Takara, A. Watanabe, Y. Sone, T. Tanaka, and A. Hirano, "Distance-adaptive spectrum resource allocation in spectrum-sliced elastic optical path network," IEEE Commun. Mag. **48**(8), 138–145 (2010).
26. M. Fayez, I. Katib, G. N. Rouskas, T. F. Gharib, and H. Faheem, "A scalable solution to network design problems: decomposition with exhaustive routing search," in *Proceedings of IEEE GLOBECOM*, December 2020.
27. G. N. Rouskas and C. Bandikatla, "Parameterized exhaustive routing with first fit for RSA problem variants," in *Proceedings of IEEE GLOBECOM*, December 2021.
28. G. N. Rouskas, S. Gupta, and P. Sharma, "Experimental evaluation of a symmetry-free parallel algorithm for spectrum allocation," in *Proceedings of IEEE GLOBECOM*, December 2022.

152.14.88.64 {ts '2023-09-11 17:02:02'}