Scalable Recursive First Fit: An Optimal Solution to the Spectrum Allocation Problem

George N. Rouskas[?†], Chaitanya Bandikatla[?]
[?]North Carolina State University, [†]King Abdulaziz University

(Invited Paper)

***Abstract*—We revisit the classical spectrum allocation (SA) problem, a fundamental subproblem in optical network design, and make three contributions. First, we show how some SA problem instances may be decomposed into smaller instances that may be solved independently without loss of optimality. Second, we prove an optimality property of the well-known first-fit (FF) heuristic. Finally, we leverage this property to develop a recursive algorithm that applies the FF heuristic to find an optimal solution efficiently. This recursive first-fit (Rec-FF) algorithm complements our recent algorithm that recursively searches the routing space, and may be combined with it to solve large routing and spectrum allocation (RSA) problem instances to optimality.**

I. Introduction

Spectrum/wavelength allocation (SA/WA) is a problem underlying a range of optical network design problems [1], including routing and wavelength allocation (RWA) [2]–[5], routing and spectrum allocation (RSA) [6], [7], traffic grooming [8], [9], and network survivability [10]. The SA and WA problems are NP-hard in networks of general topology [11]. Consequently, since the early days of optical network research a wide range of heuristic algorithms have been developed, including first-fit, best-fit, most-used, and least-loaded [12], to select which wavelength or spectrum slots to assign to each traffic demand. These heuristics represent a variety of design choices in terms of algorithmic complexity and the amount of network state information considered. First-fit, one of the earliest and simplest heuristics that requires no global knowledge, has been shown to perform well across various network topologies and sets of traffic demands [2], [13], and is one of the most commonly used algorithms for spectrum/wavelength assignment.

Motivated by the observation that many network design problems encompass two tasks, routing and resource allocation, recently we have shown [14] that it is possible to optimally decouple these two aspects and tackle each separately. Accordingly, we developed a recursive algorithm to search the routing space exhaustively yet effciently. This work complements our earlier results in [14] by developing an optimal recursive algorithm for the spectrum allocation problem.

The remainder of the paper is organized as follows. In Section II, we define the SA problem we consider in this work and show how large problem instances may be decomposed optimally into smaller ones. In Section III we prove an optimality property of the FF heuristic, and in Section IV we leverage this property to develop an optimal recursive algorithm for the SA problem. We evaluate the algorithm in Section V, and we conclude the paper in Section VI.

This work was supported by the National Science Foundation under Grant CNS-1907142.

978-3-903176-33-1 © 2021 IFIP

II. The Spectrum Allocation (SA) Problem

We consider an optical network with a topology described by graph $G = (V, A)$, where V is the set of vertices (nodes) and A is the set of arcs (directed fiber links) in the network. Let $N = |V|$ be the number of nodes and $L = |A|$ be the number of directed links; without loss of generality, we assume that if there is a fiber link from some node A to some other node B in the network, then there is a fiber link in the opposite direction, from node B to node A. We are given a set $T = \{T_i\}$ of traffic requests, and each request is a tuple $T_i = (s_i, d_i, p_i, t_i)$, where:

- s_i and d_i are the source and destination nodes, respectively, of the request,
- p_i is the (fixed and pre-determined) physical path between nodes s_i and d_i in the network over which the request must be routed, and
- t_i is the amount of spectrum (e.g., in units of spectrum slots) required to carry the traffic from s_i to d_i.

We consider the following basic definition of the spectrum allocation (SA) problem:

Definition 2.1 (SA): Given a graph $G = (V, A)$ and a set $T = \{T_i = (s_i, d_i, p_i, t_i)\}$ of traffic requests, assign t_i spectrum slots along the physical path p_i for each request T_i so as to minimize the total amount of spectrum used on any link in the network, under three constraints: 1) each request T_i is assigned a block of t_i contiguous spectrum slots (contiguity constraint), 2) each request is assigned the same block of spectrum slots along all links of its path p_i (spectrum continuity constraint), and 3) requests whose paths share a link are assigned nonoverlapping spectrum slots (nonoverlapping spectrum constraint).

In earlier work [11] we have shown that the SA problem is NP-hard even for chain (i.e., single-path) networks with four or more links. When all the spectrum demands are equal, i.e., $t_i = t \ \forall i$, the SA problem reduces to the wavelength allocation (WA) problem that can be solved in polynomial time for chain networks but remains NP-hard for rings or networks of a general topology [15]. In the next subsection, we show that under certain conditions, a large SA problem instance may be decomposed into smaller instances that can be solved independently.

Algorithm 1 Request Partition Algorithm

Input:

$G = (V, A)$: network topology

$T = \{T_i = (s_i, d_i, p_i, t_i\}$: set of traffic requests

Output:

A partition of T into subsets that pairwise use disjoint sets of links

1: {Make a singleton set for each link}
2: **for** each link $l_j \in A$ **do**
3: $\grave{}_j = \{l_j\}$;
4: **end for**
5: **for** each $T_i \in T$ **do**
6: {include all links of the path into the same up-tree, i.e., link subset}
7: F_i ;
8: **for** each $l_j \in p_i$ **do**
9: F_i $Union(F_i, Find(l_j))$
10: **end for**
11: **end for**
12: **for** each distinct non-empty subset F_i **do**
13: **return** the set of requests with paths using links in F_i
14: **end for**

A. Exact Decomposition

Consider a request set T that can be partitioned into, say, two sets T_1 and T_2, such that the paths of requests in T_1 use only links in set $A_1 \dashrightarrow A$, the paths of requests in T_2 use only links in set $A_2 \dashrightarrow A$, and the corresponding link sets are disjoint, i.e., $A_1 \setminus A_2 = ;$. In this case, it can be seen that allocation of spectrum to requests in T_1 does not affect the allocation of spectrum to requests in T_2, and vice versa. Therefore, the original SA problem on set T is decomposed exactly into two smaller SA instances on request sets T_1 and T_2, respectively, that may be solved independently; the solution to the original problem is simply the maximum of the solutions to the two smaller instances.

Algorithm 1 uses up-tree structures and *Union-Find* operations [16] to partition the set T of requests into subsets whose paths use pairwise disjoint sets of links. The algorithm starts by creating singleton sets $\grave{}_j$, each consisting of one network link $l_j \in A$. The algorithm then considers the requests in T one by one, in arbitrary order. For each request T_i, it performs a *Find* operation on each link l_j of the path p_i of T_i to locate the up-tree to which link l_j belongs; initially, the up-tree is the singleton set $\grave{}_j$. Then, the algorithm forms the *Union* of the up-trees to which the links of T_i belong. As a result, at the end of Line 11, the non-empty up-trees represent a partition of the link set A such that each link subset (i.e., up-tree) corresponds to a subset of the request set T whose paths only use links in that up-tree.

Each *Union* operation takes $O(1)$ time while each *Find* operation takes time that is logarithmic in the number of singleton sets [16], which in this case is equal to the number L of links. Therefore, the computational complexity of Algorithm 1 is determined by the **for** loop in Lines 5-11, and is $O(KL\log(L))$, where K is the number of requests in T and L is the number of links in the network.

Without loss of generality, in the remainder of this paper we assume that the request set T cannot be further decomposed into smaller independent request sets using Algorithm 1.

III. The Optimality Property of the FF Heuristic

Consider the SA problem on graph G and request set $T = \{T_i, i = 1, \cdots, K\}$. Let P be a permutation (i.e., an ordering) of the traffic requests T_i. We say that P is a *partial* (respectively, *complete*) permutation if only a subset of (respectively, all K) requests in T appear in P. Let $SOL(P)$ denote the solution to the SA problem obtained by the FF heuristic by considering each traffic request in the order implied by the permutation P. If P is a complete permutation, then $SOL(P)$ is a feasible solution to the SA problem, but if P is a partial permutation, then $SOL(P)$ is only a partial solution to the SA problem.

Let OPT denote the objective value of an optimal solution to the SA problem. A lower bound LB on the optimal objective value may be obtained by ignoring any spectrum fragmentation that may result from enforcing the spectrum contiguity and continuity constraints, and simply accounting for the fact that each link $l \in A$ must use at least as many spectrum slots as to carry all the traffic demands whose path includes this link:

$$LB = \max_{l \in A} \left\{ \sum_{T_i \in T : l \in p_i} t_i \right\} \quad (1)$$

Clearly, for any complete permutation P of the traffic requests we have that:

$$LB \leq OPT \leq SOL(P). \quad (2)$$

We now prove the following optimality property of the FF heuristic with respect to the SA problem.

Lemma 3.1 (FF Optimality Property): There exists a permutation P^*_{FF} of the traffic requests such that applying the FF heuristic to the requests in the order implied by P^*_{FF} yields an optimal solution to the SA problem, i.e., $SOL(P^*_{FF}) = OPT$.

Proof. By construction.

Consider an optimal solution to the SA problem with objective value equal to OPT. Label the slots on each link as $1, 2, \ldots, OPT$. By definition, the optimal solution is a feasible solution that satisfies all three constraints of the SA problem in that each request T_i is allocated the same block of t_i contiguous spectrum slots on each link along its path p_i, and no other request whose path shares a link with p_i is allocated slots from the same block. Let also f_i denote the slot with the lowest index within the block of t_i slots allocated to request T_i.

Let P^* be the complete permutation in which the requests T_i are listed in increasing order of f_i in the optimal solution, with ties broken arbitrarily. Consider the block of t_j contiguous spectrum slots allocated to some request T_j by the optimal solution starting at slot f_j. Let us remove this block of t_j slots from the optimal solution. In the remaining partial solution,

it is possible that there exists a block of t_j slots that start at a lower indexed slot $f_j^{\dagger} < f_j$ that are available on all links of path p_j. If so, we can allocate the lower-indexed t_j slots starting with slot $f_j^{\dagger}$ to request T_j without affecting the optimality of the solution.

Based on this observation, we modify the optimal solution by considering the requests one by one, in increasing order of f_i as listed in permutation $P^?$. For each request T_i, we remove its block of spectrum slots that starts at slot f_j from the solution, and we allocate to it an equal block of slots starting at the lowest possible slot index $f_j^{\dagger}$ in the partial solution, keeping in mind that $f_j^{\dagger}$ may be equal to f_j. This modified solution must not use more than OPT slots on any link, since any modifications involve the allocation of lower-indexed spectrum slots to requests. At the same time, since the starting solution is an optimal one, the modified solution may not use fewer than OPT slots on any link. Hence, the modified solution is an optimal one with objective value equal to OPT. Importantly, by construction the modified solution is such that no request may be allocated to a spectrum block that starts at a lower-indexed slot.

Let $P_{FF}^?$ be the complete permutation in which the requests are relabeled so that they are listed in increasing order of $f_i^{\dagger}$ in the modified solution, and let us apply the FF heuristic to this permutation. The FF heuristic allocates to each request T_i a block of t_i contiguous slots starting at the lowest-indexed slot for which such a block is available on all links of path p_i. Therefore, the FF heuristic will construct an optimal solution that is identical to the modified solution above. ■

This FF optimality property helps explain how so many studies of the SA and WA problems have found the FF heuristic to perform quite well in diverse problem instances. However, Lemma 3.1 constructs a permutation $P_{FF}^?$ on which the FF is optimal, but does so by modifying an *unknown* optimal solution and hence $P_{FF}^?$ is itself unknown. Nevertheless, the FF optimality property suggests a procedure for finding $P_{FF}^?$: enumerate all permutations of requests, run the FF heuristic on each permutation, and select the one with the smallest objective value. Assuming there is traffic between all node pairs, the size K of the request set is $O(N^2)$, where N is the number of nodes. Therefore, any algorithm that considers all possible permutations of requests to determine the optimal spectrum allocation must take time that is exponential in the size of the network, $O(N^2!)$.

In the following section we present a recursive procedure for searching efficiently the space of request permutations to determine this optimal solution.

IV. Scalable Recursive First Fit

We have developed a scalable branch-and-bound recursive first-fit (Rec-FF) procedure, shown as Algorithm 2, to search the entire space of request permutations. We start with a complete permutation P_{init} in which the K traffic requests $T_i, i = 1, \cdots, K$, are listed in decreasing order of spectrum demand t_i, and requests with the same demand are listed in decreasing order of path length. We calculate the lower bound LB on the optimal solution OPT using expression (1), and also run the FF heuristic on P_{init} to obtain an initial feasible solution $SOL(P_{init})$ which represents an upper bound on OPT. The algorithm maintains variable $BestSOL$ that indicates the best solution it has found so far; this variable is initialized as $BestSOL = SOL(P_{init})$. Although the recursive procedure will work with any initial complete permutation of requests, our earlier work and other related studies [2], [13] indicate that applying the FF heuristic to the requests in the order determined by P_{init} yields better (i.e., lower) solutions that leave a relatively small gap between LB and $SOL(P_{init})$. The Rec-FF procedure then searches the permutation space to find the permutation that yields an optimal solution, as Lemma 3.1 suggests.

Each recursive call takes two arguments: a tentative permutation P and a $Start$ index. The recursion builds permutations by maintaining a $Start$ index that takes the values $1, 2, \cdots, K$, and divides an input permutation in two parts: a *finalized* leading sub-permutation (prefix) for which the order of requests will not be modified in subsequent recursive calls, and a *tentative* trailing sub-permutation (suffix) for which the order of requests is subject to change and will be finalized by later recursive calls. The $Start$ index indicates the start of this trailing sub-permutation. Initially, the ordering of all requests is tentative, and hence the leading sub-permutation is null and all K requests belong to the trailing sub-permutation. Accordingly, the first call to Rec-FF is with $Start = 1$.

The main recursion is the **for** loop in Lines 10-22 of Algorithm 2. Essentially, the **for** loop swaps the first request of the trailing sub-permutation (i.e., the request at index $Start$) with all requests in this trailing sub-permutation, including itself (i.e., requests at index $k = Start, \cdots, K$). After making the swap for one value of k, the procedure updates the permutation (Line 16), and increments the $Start$ index (Line 17) to indicate that the leading sub-permutation of requests whose order has been finalized has increased in size by one. It then makes a recursive call (Line 18) to continue swapping requests of the trailing sub-permutation (which has decreased by one). These recursive calls, if allowed to continue without any restriction, will enumerate all possible $K!$ permutations of the K requests.

However, not all permutations will lead to a solution that is better than the currently best known one, $BestSOL$. Therefore, after making a swap and before making a recursive call, in Line 14 the algorithm runs the FF heuristic on the leading sub-permutation as it has been expanded after the swap, and compares the result to $BestSOL$. If the result is equal to or higher than $BestSOL$, then it is clear that including more requests to this sub-permutation will produce solutions that are no better than the best one found so far. In other words, continuing further down this subtree of the permutation space is not productive in terms of finding an optimal solution, and this part of the search space can be safely eliminated. Consequently, as shown in Lines 15-19, a recursive call is made only if the FF solution on this leading sub-permutation is strictly lower than $BestSOL$.

The base case for the recursion is when the order of all K requests in an input permutation P has been finalized. A complete finalized permutation is indicated whenever the input

Algorithm 2 Recursive First Fit (Rec-FF)

Input:

$G = (V, A)$: network topology
$T = \{T_i = (s_i, d_i, p_i, t_i\}$: set of traffic requests
$K = |T|$: number of traffic requests
P_{init}: initial permutation as discussed in Section IV
LB: the lower bound from expression (1)
$BestSOL$: best solution so far, initialized to $SOL(P_{init})$
$BestP$: best permutation so far, initialized to P_{init}

Output:

Best permutation and corresponding SA solution

Rec-FF($P, Start$)
P : permutation (initial call with $P = P_{init}$)
$Start$: start index of trailing sub-permutation of P that has not been finalized (initial call with $Start = 1$)

```
    {Base Case: All K requests finalized in P}
    if Start > K then
2:    S ← SOL(P); {solution obtained by FF on P}
      if S < BestSOL then {Update best known solution}
4:      BestSOL = S; BestP = P;
      end if
6:    return;
    end if
8:  {Main Recursion}
    {Swap P[Start] with all requests that follow it in P}
10: for k = Start; k ≤ K; k++ do
      Swap P[Start] with P[k];
12:   P_lead ← leading permutation P[1] ··· P[Start];
      {All requests in P_lead have been finalized}
14:   leadSOL ← SOL(P_lead);
      if leadSOL < BestSOL then
16:     newP ← permutation after the swap at Line 11;
        newStart ← Start + 1;
18:     Rec-FF(newP, newStart);
      end if
20:   {Restore P and proceed to swap the next request}
      Swap P[Start] with P[k];
22: end for
```

index $Start > K$, and this case is handled in Lines 1-7 of the algorithm. Specifically, the algorithm runs the FF heuristic on P, and if the solution is strictly better than the best known solution, the best solution is appropriately updated in Line 4, before the call returns.

Finally, we note that the Rec-FF algorithm builds a finalized permutation one request at a time. Therefore, when it invokes the FF heuristic in Line 14 on the leading sub-permutation P_{lead}, it is not necessary to run the heuristic on the entire sub-permutation. With appropriate bookkeeping (omitted from Algorithm 2 for the sake of clarity and brevity), it is only necessary to use FF to allocate spectrum for just the most recent request added to the leading sub-permutation in Line 11 by building upon the solution created by the calling function. Similarly, Line 2 of the algorithm does not actually need to run the FF heuristic at all, it can simply reuse the solution of the calling function which finalized the complete input permutation P. This optimization eliminates unnecessary computations and significantly speeds up the running time of the recursion.

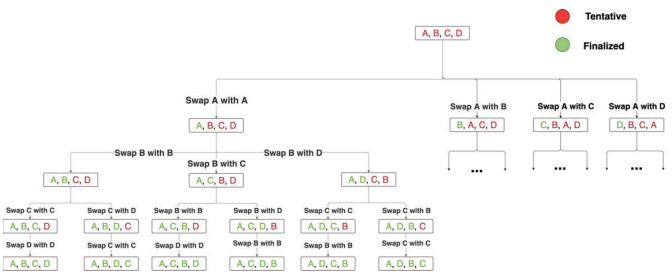

Fig. 1. Operation of the Rec-FF procedure on the set of four requests. The root of the tree represents the initial call with $P = \{A, B, C, D\}$ and $Start = 1$.

To illustrate the operation of the Rec-FF procedure, consider a set of four requests, $\{A, B, C, D\}$. Figure 1 shows part of the tree of recursive calls made, with the root of the entire tree representing the initial call with arguments $P = \{A, B, C, D\}$ and $Start = 1$. The figure is generated by assuming that the **if** condition in Line 15 of the algorithm is not checked, and hence all recursive calls are made to generate all $4! = 24$ possible permutations of requests. Also, we use red color to indicate the requests in the tentative trailing sub-permutation, and green color to indicate the requests in the finalized leading sub-permutation whose order has been set. In the initial call, all four requests are tentative (and are colored red), and the **for** loop in Lines 10-22 runs four times, each time swapping the first request A of P with each of the four requests in the set, A, B, C, and D, as indicated in the figure. The first of these recursive calls is the root of the leftmost subtree and swaps A with itself; at that point, the order of A becomes fixed (indicated in the figure by a change of color from red to green) and does not change for the remaining recursive calls in this leftmost subtree. The **for** loop of the call representing the root of the leftmost subtree ($Start = 2$) runs three times, each time swapping the second request B of the permutation passed to it with each of the three requests B, C, and D in the trailing tentative sub-sequence. This continues recursively until the six leaves of this leftmost sub-tree are reached, each representing one of the six possible permutations with request A in the first position of the permutation. The subtrees of the other three children of the root are omitted from the figure, but are similar in that they generate all 18 permutations with B, C or D in the leftmost position. For instance, the second of the recursive calls from the root of the whole tree swaps the first request A of P with request B. Subsequent calls in this subtree swap the second request with one of A, C, and D, as before, and so on, until all six permutations with B in the leftmost position are generated.

We emphasize that, in the worst case, the Rec-FF procedure may be forced to generate all, or close to all, possible permutations of requests and hence take exponential time to complete.

V. Simulation Study

We now present the results of simulation experiments to evaluate the performance of the Rec-FF algorithm on two network topologies, the 14-node, 21-link NSFNET and the 32-node, 54-link GEANT2 network, with shortest-path routing. For each topology, we create SA problem instances by generating traffic requests between all node pairs in the network as follows. We consider data rates of 10, 40, 100, 400, and 1000 Gbps. For a given problem instance, we generate a random value for the demand between a pair of nodes based on one of three distributions: 1) *Uniform*: each of the five rates is selected with equal probability; 2) *Skewed low*: the rates above are selected with probability 0.30, 0.25, 0.20, 0.15, and 0.10, respectively; or 3) *Skewed high*: the five rates are selected with probability 0.10, 0.15, 0.20, 0.25, and 0.30, respectively. Once the traffic rates between each node pair have been generated, we calculate the corresponding spectrum slots by assuming that the slot width is 12.5 GHz, and adopting the parameters of [17] to determine the number of spectrum slots that each demand requires based on its data rate and path length.

The performance measure we consider is the maximum number of spectrum slots on any network link as obtained by either the FF or Rec-FF algorithms. We let the Rec-FF algorithm run until it either reaches the lower bound (in which case we know for certain it has found an optimal solution) or it reaches a 5-hour limit on running time; while in the latter case we are not certain that the algorithm has found an optimal solution, as we discuss shortly, we believe that the solution is very close to optimal. For meaningful comparisons between problem instances, we normalize the solutions returned by FF or Rec-FF by dividing with the lower bound LB for the corresponding instance from expression (1). Clearly, the closer the normalized value is to 1.0, the better the solution.

Figures 2 and 3 present results for the NSFNET and GEANT2 topologies, respectively. Each figure includes three subfigures, one each for demand matrices generated by the skewed low, skewed high, and uniform distributions, respectively. Each subfigure plots the normalized FF solution, the normalized Rec-FF solution, and the normalized lower bound (the last one as a horizontal line at $y = 1.0$), for each of 100 random problem instances generated for the stated parameters (i.e., network topology and traffic demand distribution).

We first note that the FF algorithm produces solutions of good quality that are within 30% (respectively, 12%) of the lower bound for the 300 NSFNET (respectively, GEANT2) problem instances. These results are consistent with earlier research indicating that the FF algorithm performs well. Regarding the Rec-FF algorithm, we observe that it finds better solutions than FF in most instances. Table I summarizes the average relative performance of the FF and Rec-FF algorithms in terms of how far (in percentage) terms their solutions are from the lower bound, the number of instances (out of 100 for each distribution) that the Rec-FF produces better solutions than FF, the number of instances that Rec-FF finds a solution equal to the lower bound (i.e., a guranteed optimal solution), and the average absolute difference between the FF and Rec-FF solutions, in spectrum slots. For the NSFNET (respectively,

Fig. 2. Normalized solutions to 300 problem instances, NSFNET

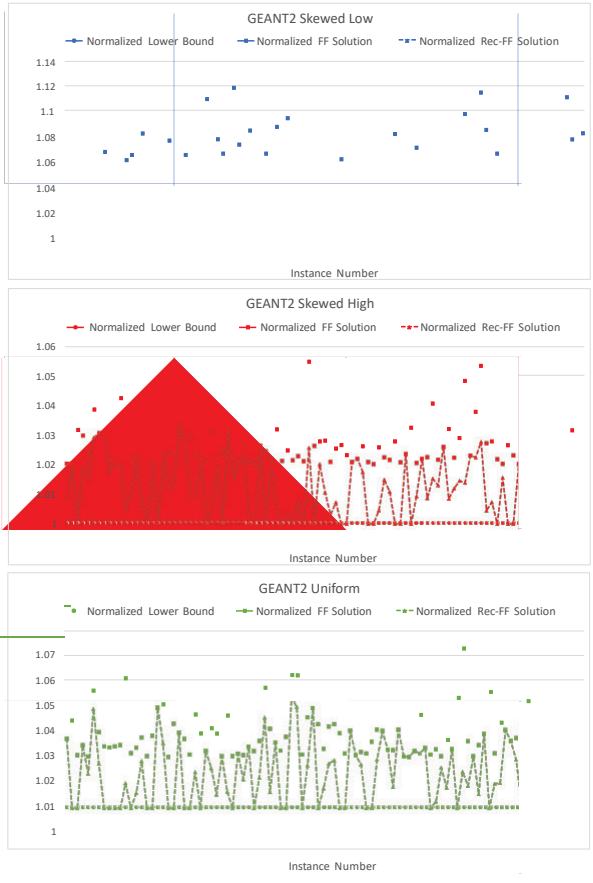

Fig. 3. Normalized solutions to 300 problem instances, GEANT2

TABLE I
RELATIVE PERFORMANCE OF FF AND REC-FF ALGORITHMS

	Skewed	9.28%	5.46%	53	20	3.78

GEANT2) network, Rec-FF improves on the FF solution in 47-53 (respectively, 71-79) instances, depending on the traffic distribution, of which it finds a solution equal to the lower bound in 20-26 (respectively, 14-33) instances. Also, although the percentage improvement over the FF solution is lower for the GEANT2 network, the absolute difference is more than twice that for the NSFNET network. In other words, even a small improvement in the larger GEANT2 network results in significantly larger spectrum savings, especially since it applies across many more network links.

Finally, Figure 4 shows the improvement in the solutions found by the Rec-FF algorithm as a function of how long the algorithm has run, starting from the FF solution it receives as input at time $t = 0$ until we terminate the algorithm after 5 hours (note that the time axis is not in linear scale). We show two instances, one for NSFNET and one for GEANT2, for which Rec-FF finds a solution that is better than FF but is higher than the lower bound (hence the algorithm runs for the full 5 hours). It takes less than 5 sec (respectively, 45 sec) for Rec-FF to find the best solution in the case of NSFNET (respectively, GEANT2); in the remaining time the algorithm explors solutions that are not better than the best one found in the first few seconds. These trends are very similar to the ones we observed for all instances of the corresponding networks, and indicate that 1) it takes only a few seconds for Rec-FF to find its best solution, and 2) even if this solution is not optimal, it is likely very close to optimal.

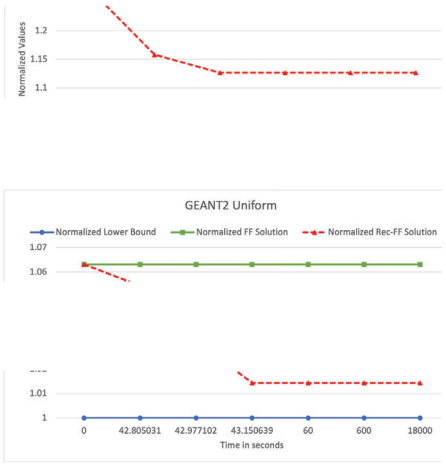

Fig. 4. Improvement of Rec-FF solution vs. time

VI. CONCLUDING REMARKS

We have developed Rec-FF, an algorithm that applies the FF heuristic recursively to solve optimally the SA problem. The algorithm generally takes less than one minute to produce solutions that are very close to the lower bound and which, we conjecture, are optimal. We plan to integrate Rec-FF with the algorithm in [14] so as to solve large RSA problems efficiently.

REFERENCES

[1] J. M. Simmons, *Optical Network Design and Planning*. Springer, 2008.

[2] J. Simmons and G. N. Rouskas, "Routing and wavelength (spectrum) allocation," in *B. Mukherjee, I. Tomkos, M. Tornatore, P. Winzer, and Y. Zhao (Editors), Springer Handbook of Optical Networks*, 2020.

[3] G. N. Rouskas, "Routing and wavelength assignment in optical WDM networks," in *J. Proakis (Editor), Wiley Encyclopedia of Telecommunications*, John Wiley & Sons, 2001.

[4] R. Dutta and G. N. Rouskas, "A survey of virtual topology design algorithms for wavelength routed optical networks," *Optical Networks*, vol. 1, pp. 73–89, January 2000.

[5] B. Jaumard, *et al.*"Comparison of ILP formulations for the RWA problem," *Optical Switch. & Netw.*, vol. 3-4, pp. 157–172, 2007.

[6] M. Klinkowski, P. Lechowicz, and K. Walkowiak, "Survey of resource allocation schemes and algorithms in spectrally-spatially flexible optical networking," *Optical Switch. & Netw.*, vol. 27, no. C, pp. 58–78, 2018.

[7] S. Talebi, F. Alam, I. Katib, M. Khamis, R. Khalifah, and G. N. Rouskas, "Spectrum management techniques for elastic optical networks: A survey," *Optical Switc. & Netw.*, vol. 13, pp. 34–48, July 2014.

[8] R. Dutta and G. N. Rouskas, "Traffic grooming in WDM networks: Past and future," *IEEE Network*, vol. 16, pp. 46–56, Nov/Dec 2002.

[9] H. Wang and G. N. Rouskas, "Hierarchical traffic grooming: A tutorial," *Computer Networks*, vol. 69, pp. 147–156, August 2014.

[10] D. Zhou and S. Subramaniam, "Survivability in optical networks," *IEEE Network*, vol. 14, pp. 16–23, November/December 2000.

[11] S. Talebi, E. Bampis, G. Lucarelli, I. Katib, and G. N. Rouskas, "Spectrum assignment in optical networks: A multiprocessor scheduling perspective," *J. Optical Comm. & Netw.*, vol. 6, pp. 754–763, Aug 2014.

[12] H. Zang, J. P. Jue, and B. Mukherjee, "A review of routing and wavelength assignment approaches for wavelength-routed optical WDM networks," *Optical Networks*, vol. 1, pp. 47–60, January 2000.

[13] Y. Zhu, G. N. Rouskas, and H. G. Perros, "A comparison of allocation policies in wavelength routing networks," *Photonic Network Communications*, vol. 2, pp. 265–293, August 2000.

[14] 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," December 2020.

[15] S. Huang, R. Dutta, and G. N. Rouskas, "Traffic grooming in path, star, and tree networks: Complexity, bounds, and algorithms," *IEEE Journal on Selected Areas in Communications*, vol. 24, pp. 66–82, April 2006.

[16] M. T. Goodrich and R. Tamassia, *Data Structures & Algorithms in Java*. New York: John Wiley & Sons, 2010.

[17] M. Jinno, *et al.*, "Distance-adaptive spectrum resource allocation in spectrum-sliced elastic optical path network," *IEEE Communications Magazine*, vol. 48, no. 8, pp. 138–145, 2010.