Experimental Evaluation of a Symmetry-Free Parallel Algorithm for Spectrum Allocation

George N. Rouskas, Shubham Gupta, Priya Sharma
North Carolina State University

GLOBECOM 2022 - 2022 IEEE Global Communications Conference | 978-1-6654-3540-6/22/$31.00 ©2022 IEEE | DOI: 10.1109/GLOBECOM48099.2022.10001401

***Abstract*—Recursive First Fit (RFF) is an optimal algorithm for the offline spectrum allocation (SA) problem that we developed recently [1]. To the best of our knowledge, RFF is the first algorithm for networks of general topology that is spectrum symmetry-free, i.e., it does not consider any equivalent solutions that are the result of spectrum slot permutations. The algorithm applies the first-fit (FF) heuristic to solve the SA problem, and hence it can be readily implemented. 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 the RFF algorithm explores a vast number of symmetry-free solutions and, for moderate-size networks, it takes mere seconds to yield solutions that are either optimal or very close to the lower bound.**

I. Introduction

Spectrum allocation (SA) is integral to optical network design and has been studied extensively for decades in a variety of contexts, usually coupled to other objectives including routing [2]–[4], traffic grooming [5], network survivability [6], and virtual topology design [7], [8]. The SA problem is known to be intractable even when considered in isolation, i.e., separately from other aspects of network design [9]. Yet spectrum symmetry, an aspect unique to SA, makes the problem especially challenging to optimal algorithms, including integer linear programming (ILP) solvers.

Spectrum symmetry arises from the fact that slices of optical spectrum are *interchangeable* [10]. Hence, by simply permuting the spectrum slots, every solution (optimal or not) to the SA problem yields an exponential number of equivalent solutions [11]. Conventional ILP solvers encompass this vast number of distinct yet equivalent solutions, and hence their running time is unnecessarily long [11]. A decade ago, we developed an ILP formulation based on maximal independent sets (MIS) [12] for the offline routing and wavelength allocation (RWA) problem in ring networks. The formulation does not suffer from the symmetry problem and can obtain optimal solutions for large 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 networks of general topology. Thus, an optimal solution approach that avoids spectrum symmetry has eluded the research community until recently.

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

The first-fit (FF) algorithm is a simple heuristic for the SA problem that has been shown to be effective across a wide range of network topologies and traffic demands [13]–[15]. Recently, we have proved an optimality property of the FF heuristic [1] that allows for the design of symmetry-free algorithms. Specifically, we showed that there exists a permutation of the traffic demands such that applying the FF heuristic to the traffic demands 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 demand permutations, and it is not necessary to account for any assignment of spectrum slots other the one determined by the FF heuristic.

Following up on this insight, we also developed recursive first-fit (RFF), an optimal branch-and-bound algorithm [1]. RFF searches the space of demand permutations and applies the FF heuristic as it incrementally builds each permutation during the search. While the demand permutation space is itself exponential in size, RFF represents a significant improvement over existing approaches as it completely sidesteps the spectrum symmetry challenge.

In this work we present two parallel implementations of the RFF algorithm and evaluate them experimentally using a comprehensive set of metrics. In Section II we introduce briefly the RFF algorithm and discuss the two parallel implementations. We discuss our experimental setup and results in Section III, and we conclude the paper in Section IV.

II. The Symmetry-Free RFF Algorithm and Its Parallel Implementation

The RFF algorithm is described in detail in [1]. For the sake of completeness, in this section we first explain the basic operation of RFF and then we present two approaches to executing the algorithm in parallel.

Consider an instance of the SA problem with K traffic requests, each request corresponding to one node-pair in the network. A request consists of a path and demand (i.e., number of spectrum slots) for the traffic between two nodes; we assume that the path between the two nodes is predetermined, i.e., a routing decision has already been made. The objective is to assign along the path of each request a block of contiguous and continuous spectrum slots of size equal to its demand, so as to minimize the index of the highest slot assigned on any link of the network.

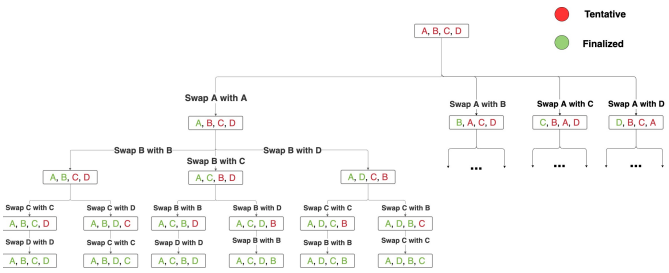

Fig. 1. Tree of RFF calls on a set of four requests. The root of the tree represents the initial call with initial permutation $P = [A, B, C, D]$.

Due to the FF optimality property we discussed earlier, an optimal solution to the SA problem can be obtained by applying the FF heuristic to all permutations of the K requests and selecting the best one. RFF [1] is a recursive branch-and-bound algorithm that searches the space of $K!$ demand permutations efficiently. RFF starts with an initial permutation, P, 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 request at a time. A recursive call also applies the FF heuristic to the partial permutation built so far. Once RFF has built a complete permutation (i.e., one consisting of some ordering of all K requests), 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 1 shows the tree of recursive calls that RFF makes on an instance with four requests, 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 procedes 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 request (A) in the permutation, and proceeds recursively to finalize the remaining three requests. The leaves of the tree represent the distinct permutations, and hence, there are $K!$ leaves. In the example of Figure 1, the leftmost subtree of the root has $3! = 6$ leaves corresponding to the six permutations in which request 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.

We also observe from Figure 1 that the sequences of recursive calls that belong to 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 reach a leaf node and 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.

Based on this discussion, the RFF algorithm represents a significant improvement over existing methods in searching for an optimal solution to the SA problem because of three features:

1) it is spectrum symmetry-free, i.e., it does not consider any equivalent solutions that are the result of spectrum slot permutations;
2) it uses indirect exploration to eliminate whole subtrees of the solution space without explicitly visiting all the permutations they contain; and
3) it can be executed in parallel.

Next, we discuss two parallel implementations of RFF.

A. Parallel Implementation of RFF

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, and hence K is $O(N^2)$, where N is the number of network nodes. Hence, for network topologies encountered in practice, we assume that $M < K$. The objective is 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, we must ensure not only that the algorithm explores the largest possible subset of the solution (permutation) space, but also

that the subset explored is representative of the entire state space and is not limited to the neighborhood around the initial permutation provided as input.

As we mentioned earlier, a recursive call of RFF that starts at the root of a subtree (refer to Figure 1) 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 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 alotted is not sufficient for RFF to visit the whole subtree, it will visit the 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 [1], 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. Finally, we note that, given our assumption that $M < K$, and often that $M \ll K$, it is generally 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 parallelization strategy 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 which differ in the depth of the RFF recursion call tree, shown in Figure 1, at which threads are deployed to execute in parallel.

Depth-0 Parallelization Strategy. Note that the subtrees of the root divide the solution space into K non-overlapping subsets ($K = 4$ in Figure 1), and hence this is a natural point to introduce parellelism. Since we assume that $M < K$, we run RFF as follows. Let $m = \lceil K/M \rceil$ and $t = T/m$. We run m batches of threads, where each batch consists of M threads (the m-th batch may have fewer than M threads) running in parallel for t units of time. The first batch consists of threads starting at the M leftmost subtrees (children) of the root. When the first batch completes after t time units, the best solution found by this batch is passed to the next batch of threads that start at next M subtrees of the root; and so on, until the last batch of threads starting at the rightmost subtrees of the root complete at time T.

Depth-1 Parallelization Strategy. While the above is a natural and straightforward parallelization strategy, the thread at each subtree of the root will run for a relatively small time and, for values of K corresponding to typical networks, it will not be able to explore the whole subtree. Based on our earlier discussions, this strategy will not explore the rightmost parts of any subtree of the root, and hence it will miss many regions of the solution space. Our second strategy introduces parallelism at depth 1 of the tree, i.e., each thread explores a subtree of a child of the root. The root has K children, each of which has $K-1$ subtrees for a total of $K(K-1)$ subtrees. Similar to the first strategy, we deploy $n = \lceil K(K-1)/M \rceil$ batches of threads, each batch running for $s = T/n$ time units. 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 for the first strategy.

III. Numerical Results

A. Simulation Setup

We have carried out a large number of 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 [16] which consists of more than 1,000 compute nodes and over 10,000 cores.

The experimental setup is similar to the one we used in our original work on RFF in [1]. Specifically, we consider two topologies, the 14-node, 21-link NSFNet and the 32-node, 54-link GEANT2 networks, and shortest path routing. We create SA problem instances by generating traffic requests 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: 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. 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 [20] to determine the number of spectrum slots that each demand requires based on its data rate and path length. For each traffic distribution, we generate 100 random problem instances.

As in [1], 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 traffic demands 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 $K = 91$ requests, and we set $T = 270$ min as the bound on the running time. Hence, for the Depth-0 strategy we deployed $m = 3$ batches of threads, each batch running for 90 min. For the Depth-1 strategy, there are $n = 90 \times 91/32 = 256$ batches, and we let each batch run

for 1 min; 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 16 batches of 32 threads; therefore, we set $T = 80$ min as the bound on running time and let each batch run for 5 min. The Depth-1 strategy, on the other hand, would need to explore $495 * 496 = 245,520$ subtrees and, hence, it would require 7,673 batches of 32 threads; each batch would then need to run for less than one second 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 only consider 80 children uniformly spaced apart, i.e., children $1, 7, 13, \cdots, 475$. For each of these children we create a batch of 32 threads, each thread exploring a subtree of the child node, and wil let each batch run for 1 min. 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. Clearly, there are various approaches for sampling the solution space at Depth-1, but considering such approaches is outside the scope of our work.

B. Results and Discussion

We evaluate the RFF algorithm using several performance measures: 1) the solution quality relative to the FF heuristic 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.

Tables I and II summarize the results we have obtained for the NSFNet and GEANT2 topologies, respectively. The tables 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 tables represents an average over 100 problem instances for the stated network, algorithm, and traffic distribution. 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.

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 [1], 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.

Tables III and IV present a different perspective regarding the quality of the solutions for the NSFNet and GEANT2

TABLE I
FF AND RFF SOLUTION QUALITY AS % FROM LB, NSFNET

Traffic	FF	RFF	
		Depth-0	Depth-1
Uniform	10.12%	4.49%	3.20%
Skewed Low	11.73%	6.01%	5.04%
Skewed High	9.28%	3.74%	2.95%

TABLE II
FF AND RFF SOLUTION QUALITY AS % FROM LB, GEANT2

Traffic	FF	RFF	
		Depth-0	Depth-1
Uniform	2.88%	0.56%	0.39%
Skewed Low	6.57%	2.15%	1.58%
Skewed High	2.66%	0.41%	0.23%

TABLE III
RFF SOLUTIONS RELATIVE TO FF AND LB, NSFNET

Traffic	# instances <FF		# instances $= LB$	
	Depth-0	Depth-1	Depth-0	Depth-1
Uniform	63	74	34	51
Skewed Low	58	64	28	44
Skewed High	77	80	41	56

TABLE IV
RFF SOLUTIONS RELATIVE TO FF AND LB, GEANT2

Traffic	# instances <FF		# instances $= LB$	
	Depth-0	Depth-1	Depth-0	Depth-1
Uniform	92	93	62	77
Skewed Low	86	86	28	50
Skewed High	93	94	51	79

TABLE V
NUMBER OF PERMUTATIONS EXPLORED, NSFNET

Traffic	Reached LB		Did not reach LB	
	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 VI
NUMBER OF PERMUTATIONS EXPLORED, GEANT2

Traffic	Reached LB		Did not reach LB	
	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

TABLE VII
TIME (SEC) TO REACH THE BEST SOLUTION, NSFNET

Traffic	Depth-0				Depth-1			
	Min	Median	Avg	Max	Min	Median	Avg	Max
Uniform	4	4	4	5	4	5	108	1,205
Skewed Low	4	4	4	5	5	5	71	664
Skewed High	4	4	10	77	4	5	98	1505

TABLE VIII
TIME (SEC) TO REACH THE BEST SOLUTION, GEANT2

Traffic	Depth-0				Depth-1			
	Min	Median	Avg	Max	Min	Median	Avg	Max
Uniform	33	42	130	1133	33	41	161	1,241
Skewed Low	34	42	54	398	41	42	216	1421
Skewed High	42	42	194	2247	33	41	198	2501

topologies, respectively. Specifically, the two tables list the number of problem instances for which the Depth-0 and Depth-1 solutions are 1) either better than the corresponding FF solution or 2) equal to the lower bound LB of the corresponding instance. To interpret the results in the two tables we note that a solution that is 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.

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 either 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.

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 that 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 V and VI 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 $91! \approx 1.35\text{E}140$ for NSFNet and $496! \approx 1.98\text{E}1123$ 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 alloted 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.

Tables VII and VIII provide insight into the running time of the algorithm for the two networks. Specifically, the tables

list 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 or 5 seconds, depending on the strategy, to find the best solution. For the Depth-1 strategy that yields lower solutions, it takes a about 25 min in the worst case to find the best solution for a handful of outlier instances. In the GEANT case, the median is around 42 sec for both strategies and all three traffic distributions, and the maximum is around 40 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 unfovorable 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 larger amount of time. The results shown in Tables VII and VIII 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., itis 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 time T can be afforded, a parallelization strategy that starts at a higher depth 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 heuristic, it can be readily implemented in production environments.

IV. Concluding Remarks

We have presented two parallelization strategies for recursive first-fit (RFF), a symmetry-free optimal algorithm for the offline spectrum allocation (SA) problem. Parallel implementations of RFF explore vast amounts of the solution space and yield solutions at or near the lower bound. We plan to extend this work to explore the potential of RFF in larger-size networks by extending the parallelization strategies we developed above, and to combine RFF with the routing algorithms we developed in [21], [22] so as to tackle large RSA problems efficiently.

References

[1] G. N. Rouskas and C. Bandikatla, "Recursive first fit: A highly parallel optimal solution to spectrum allocation." *IEEE/Optica Journal of Optical Communications and Networking*, vol. 14, pp. 165-176, April 2022.

[2] B. Jaumard, C. Meyer, and B. Thiongane, "Comparison of ILP formulations for the RWA problem," *Optical Switching and Networking*, vol. 3-4, pp. 157–172, 2007.

[3] M. Klinkowski, P. Lechowicz, and K. Walkowiak, "Survey of resource allocation schemes and algorithms in spectrally-spatially flexible optical networking," *Optical Switching & Networking*, vol. 27, pp. 58–78, 2018.

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

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

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

[7] 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.

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

[9] S. Talebi, E. Bampis, G. Lucarelli, I. Katib, and G. N. Rouskas, "Spectrum assignment in optical networks: A multiprocessor scheduling perspective," *Journal of Optical Communications and Networking*, vol. 6, pp. 754–763, August 2014.

[10] R. Ramaswami and K. Sivarajan, "Routing and wavelength assignment in all-optical networks," *IEEE/ACM Transactions on Networking*, vol. 3, pp. 489–500, October 1995.

[11] B. Jaumard, C. Meyer, and B. Thiongane., "ILP formulations for the routing and wavelength assignment problem: Symmetric systems," in *Handbook of Optimization in Telecom.*, pp. 637–677, 2006.

[12] E. Yetginer, Z. Liu, and G. N. Rouskas, "Fast exact ILP decompositions for ring RWA," *Journal of Optical Communications and Networking*, vol. 3, pp. 577–586, July 2011.

[13] 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.

[14] 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*, Springer, 2020.

[15] 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.

[16] "NC State Henry2 Linux Cluster." https://projects.ncsu.edu/hpc/About/ComputeResources.php.

[17] Z. Ortiz, G. N. Rouskas, and H. G. Perros, "Scheduling of multicast traffic in tunable-receiver WDM networks with non-negligible tuning latencies," *Proc. ACM SIGCOMM*, pp. 301–310, Sep. 1997.

[18] V. Sivaraman and G. N. Rouskas, "HiPeR-ℓ: A High Performance Reservation protocol with ℓook-ahead for broadcast WDM networks," in *Proceedings of INFOCOM '97*, pp. 1272–1279, IEEE, April 1997.

[19] L. Xu, H. G. Perros, and G. N. Rouskas, "A simulation study of optical burst switching access protocols for WDM ring networks," *Computer Networks*, vol. 41, pp. 143–160, January 2003.

[20] 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 Communications Magazine*, vol. 48, no. 8, pp. 138–145, 2010.

[21] 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," *Proc. IEEE GLOBECOM 2020*.

[22] G. N. Rouskas and C. Bandikatla, "Parameterized exhaustive routing with first fit for RSA problem variants," *Proc. IEEE GLOBECOM 2021*.