Combining a Parallel Branch-and-Bound Algorithm with a Strong Heuristic to Solve the Sequential Ordering Problem

Ghassan Shobaki
California State University, Sacramento
Sacramento, California, USA
ghassan.shobaki@csus.edu

Taspon Gonggiatgul
California State University, Sacramento
Sacramento, California, USA

Jacob Normington
California State University, Sacramento
Sacramento, California, USA

Pınar Muyan-Özçelik
California State University, Sacramento
Sacramento, California, USA

ABSTRACT

In this paper, we describe how to combine a parallel branch-and-bound (B&B) algorithm and a strong heuristic to solve the Sequential Ordering Problem (SOP), which is an NP-hard optimization problem. A parallel B&B algorithm is run in parallel with the Lin-Kernighan-Helsgaun heuristic algorithm, which is known to be one of the strongest heuristic algorithms for solving the SOP. The best solutions found by each algorithm are shared with the other algorithm, and each algorithm benefits from the better solutions found by the other. With the better solutions found by B&B, LKH can find even better solutions. With the better solutions found by LKH, B&B will have a tighter upper bound that enables it to prune at shallower tree nodes and thus complete it search faster. The combined algorithm is evaluated experimentally on the SOPLIB and TSPLIB benchmarks. The results show that the combined algorithm gives significantly better performance than any of the B&B algorithm or the LKH heuristic individually. Significant improvements in both speed and solution quality are seen on both benchmark suites. For example, the proposed algorithm delivers a geometric-mean speedup of 10.17 relative to LKH on the medium-difficulty SOPLIB instances. On the hard SOPLIB instances, it improves the cost by up to 22% relative to B&B and up to 90% relative to LKH

KEYWORDS

parallel branch-and-bound, Lin-Kernighan-Helsgaun heuristic, sequential ordering problem, combinatorial optimization

ACM Reference Format:
Ghassan Shobaki, Taspon Gonggiatgul, Jacob Normington, and Pınar Muyan-Özçelik. 2023. Combining a Parallel Branch-and-Bound Algorithm with a Strong Heuristic to Solve the Sequential Ordering Problem. In *52nd International Conference on Parallel Processing Workshops (ICPP-W 2023), August 07–10, 2023, Salt Lake City, UT, USA*. ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3605731.3608929

1 INTRODUCTION

We have recently proposed a parallel Branch-and-Bound (B&B) algorithm for the Sequential Ordering Problem (SOP) [15, 16]. The SOP is a generalization of the Traveling Salesman Problem (TSP), which is a well-known NP-hard optimization problem. Given a weighted graph and a dependence graph representing precedence constraints among the vertices, the objective in the SOP is finding a minimum-cost Hamiltonian path in the weighted graph that satisfies the precedence constraints in the dependence graph.

Our parallel B&B algorithm is based on a previously proposed sequential B&B algorithm for the SOP [22, 30, 34]. The parallel B&B algorithm delivers significant speedup relative to the sequential algorithm, but it fails to solve many hard SOPLIB and TSPLIB instances. In this paper, we show how this parallel B&B algorithm can be combined with a strong heuristic algorithm to produce significantly better results than any of the two algorithms individually.

The heuristic algorithm used in this work is the Lin-Kernighan-Helsgaun (LKH) algorithm [19–21], which has proven to be highly effective at computing optimal or near-optimal solutions to the TSP and the SOP quite fast. The main disadvantage of the LKH algorithm is that it is a heuristic-based algorithm that may not find an optimal solution.

In this work, we combine the LKH algorithm with our parallel B&B algorithm to produce a hybrid algorithm that captures the strengths of both algorithms. The strength of the B&B algorithm is that it is an exact algorithm. So, it produces provably optimal solutions if it terminates within a given time limit. The strength of the LKH heuristic, on the other hand, is that it discovers optimal or near-optimal solutions relatively quickly using a local search.

In the proposed algorithm, the best solution found by any algorithm is shared between the two algorithms, thus allowing each algorithm to benefit from the better solutions found by the other. More specifically, the strong local-search technique used in LKH to find near-optimal solutions early enables the B&B algorithm to prune earlier and thus complete its exhaustive search earlier.

The search order of a B&B algorithm is an important factor that determines its performance. With a better search order, better solutions are discovered earlier, thus enabling pruning at shallower nodes in the search tree, which leads to a faster search. Achieving a good search order in a B&B algorithm is very challenging. In the proposed algorithm, the power of LKH is used to find better solutions early, and thus enable pruning in the B&B algorithm at shallower tree nodes. Experimentally, using a strong heuristic in parallel with a B&B algorithm proved to be a more effective way of finding near-optimal solutions early than optimizing the search order of the B&B algorithm.

Our experimental evaluation using TSPLIB [31] and SOPLIB [29] shows that the combined algorithm greatly improves both the execution time and the solution quality (the cost of the final solution) relative to any of the individual algorithms. On the medium-difficulty instances, the geometric-mean speedup delivered by the combined algorithm relative to the pure B&B algorithm is 1.19 on SOPLIB and 1.39 on TSPLIB. It also delivers a geometric-mean speedup of 10.17 relative to the LKH algorithm on the medium-difficulty SOPLIB instances. On the medium-difficulty TSPLIB instances, however, the combined algorithm under-performs the LKH algorithm, because the latter performs particularly well on these instances. The combined algorithm tends to run faster than the LKH algorithm on larger instances with heavier precedence constraints.

On the hard SOPLIB and TSPLIB instances, the combined algorithm finds better solutions relative to the pure B&B algorithm for 24 instances and better solutions relative to LKH for 18 instances. The maximum cost improvement is 22% relative to B&B and 90% relative to LKH. So, the final costs computed by the combined algorithm are significantly better than those computed by either pure B&B or LKH.

Yet, the results show that more work is needed on improving the the combined algorithm's ability to prove the optimality of the best solutions found by either the B&B algorithm or the LKH algorithm. This will be the focus of our future work.

2 PROBLEM DEFINITION

An instance of the Sequential Ordering Problem (SOP) consists of a cost graph $G = (V, E)$ and a precedence graph $P = (V, R)$ defined on the same set of vertices V, as well as a start vertex s and a final vertex f that both belong to V.

The cost graph is a complete weighted directed graph in which each edge (i, j) in E is assigned a weight $w(i, j)$. A path in the graph is a sequence of edges from E. The cost of a path is the sum of the weights of the edges that constitute that path. A Hamiltonian path is a path that visits *every* vertex in the graph exactly once. A Hamiltonian path is guaranteed to exist in a complete graph.

The precedence graph P is a directed graph in which an edge (x, y) in R indicates that vertex x must appear before vertex y in any *feasible path*. If the precedence constraints in P imply that vertex n cannot appear immediately after vertex m in any feasible path, the weight of edge (m, n) in G will be irrelevant, and we follow the convention of setting this weight to -1.

The SOP is the problem of finding a minimum-cost Hamiltonian path in G that starts with s, ends with f and satisfies the precedence constraints imposed by P.

3 PREVIOUS WORK

Since its introduction in 1988 [9], several sequential and parallel algorithms have been proposed to solve the SOP, also known as the Precedence-Constrained Traveling Salesman Problem. Sequential approaches utilize different heuristic techniques, including particle swarm optimization [1], ant colony optimization [11, 35] and the Lin-Kernighan-Helsgaun algorithm [21]. Various exact methods have also been proposed, including the cutting plane [2, 17], the Lagrangian relaxation [10], dynamic programming [27, 32], branch-and-cut [3, 18], branch-and-bound [22, 28, 34] and constraint programming [23].

Different parallel B&B approaches have been proposed for solving different optimization problems [6, 12, 13]. However, exact parallel algorithms for the SOP are understudied. The only published algorithms that we are aware of in this area are the algorithm of Salii and Sheka [33] and our parallel B&B algorithm [15, 16].

Various kinds of parallel B&B algorithms have been used to solve other combinatorial optimization problems, such as the Traveling Salesman[8], Quadratic Assignment [6, 13], Knapsack [24], Maximum-Clique [26], Flow-Shop Scheduling [5, 13, 14], N-Queens [13], Blocking Job Shop Scheduling [7] and Optimal Batch Plants Design [4]. These algorithms utilize different parallel architectures, including processor networks [8], GPUs [4, 24], multi-core CPUs [4, 14, 26], clusters [4] and hybrid platforms that use a combination of GPUs and multi-core CPUs [5, 7, 13].

The SOP is similar to other permutation-based optimization problems like the TSP and Flow-Shop Scheduling. However, it is more complex than these problems since it involves precedence constraints, which make developing a parallel B&B algorithm more challenging, due to harder load balancing and load estimation.

4 ALGORITHM DESCRIPTION

In this section, we first summarize the parallel B&B algorithm and the LKH algorithm, and then we describe the combined algorithm proposed in this paper. The details of the B&B algorithm and the LKH algorithm may be found in the original papers that we cite below. They are only summarized here.

4.1 Sequential B&B Algorithm

The parallel B&B algorithm is based on the sequential B&B algorithm that was originally proposed by Shobaki and Jamal [34] and later enhanced by Jamal et al. [22]. The enhanced algorithm uses a lower bound (LB) that is based on relaxing the SOP into a Minimum-Cost Perfect Matching (MCPM) problem.

The B&B algorithm exhaustively explores the solution space by constructing an *enumeration tree*. Each leaf in the tree represents a complete feasible solution, and each internal node represents a partial solution. A solution is constructed incrementally by adding one vertex at a time to the current partial solution.

At each tree node, a partial path has been constructed by selecting a sequence of vertices. The sub-problem to be solved at that node is finding the optimal order for the remaining vertices. The LBs of all possible next vertices at that node are computed and the vertex with the lowest LB is added to the current partial path. The feasible solutions in a sub-problem's solution space (sub-space) are the leaves of the sub-tree below that node.

To speed up the search, pruning techniques are applied at each node. If a pruning technique indicates that no better solution than the current best solution can be found below the current node, the algorithm backtracks to the previous node, thus pruning the sub-tree below the current node.

The two pruning techniques used in the sequential B&B algorithm are history-based domination and the MCPM LB. History domination stores information about previously visited nodes in a history table and then uses them to quickly process all the similar nodes that are visited later [34].

The second pruning technique is relaxing the SOP into a MCPM problem and then solving it using the dynamic Hungarian algorithm [22]. History domination is always applied before the dynamic Hungarian algorithm, because it is much faster.

4.2 The Parallel B&B Algorithm

Our recently-proposed parallel B&B algorithm is a pool-based algorithm that uses asynchronous multi-threading [16]. It consists of a collection of techniques that are designed to effectively search the solution space using multiple parallel threads. These techniques include thread restart, parallel history-based domination, history-table memory management, a global-pool assignment technique and a work-stealing technique for load balancing. The details may be found in the original paper [16].

In the parallel B&B algorithm, Breadth-First-Search (BFS) is initially used to split the problem into smaller sub-problems that are stored in a global pool. The sub-problems (tree nodes) in the global pool are then assigned to threads. Each thread explores the sub-tree below its assigned tree node as in the sequential algorithm. When a thread completes exploring its assigned tree node, it is assigned a new node from the global pool. If the global pool is empty, a thread that has completed exploring its assigned node will steal part of the load of an active thread [16].

4.3 The LKH Algorithm

The LKH algorithm [19, 20] is an extension of the original Lin-Kernighan (LK) algorithm [25]. It is a local-search algorithm in which better solutions are found by applying edge-exchange operations to a given solution. Exchanges continue until a certain termination condition is reached. The algorithm does not prove optimality unless it finds a solution at a pre-computed lower bound. In the TSP, a solution is a tour, which is a simple cycle in which every vertex appears exactly once.

Given a feasible but non-optimal solution S, a better solution S' can be constructed by changing k edges $x_1, x_2, x_3 \dots x_k$ in S. A tour is said to be k-optimal (or k-opt for short) if it is impossible to obtain a better solution (a shorter tour) by replacing k of its edges with other edges. The number of operations to test all k-exchanges increases rapidly as the number of cities increases. Thus, the values $k = 2$ and $k = 3$ are the most commonly used values in practice. The LKH algorithm is based on a variable k-opt where the k value is changed in each iteration until a termination condition is reached.

The LKH algorithm repeatedly performs edge exchanges that reduce the length of the tour until it finds a k-opt tour. Since its initial introduction, many updates have been made to the LKH algorithm to further enhance its performance and extend it to solve more complex problems. The latest LKH algorithm (LKH3) added support for precedence-constrained problems, such as the SOP [21].

4.4 The Combined Algorithm

In the combined algorithm proposed in the current paper, the LKH algorithm is run in parallel with our parallel B&B algorithm. One thread is used to run the LKH algorithm, while the rest of the threads run the parallel B&B algorithm. In the experimental evaluation, 32 threads are used. One thread runs LKH, and the remaining 31 threads run B&B.

The best solution found so far and its cost are shared by the B&B threads and the LKH thread. Both the B&B algorithm and the LKH algorithm benefit from this shared best solution. Whenever a better solution is found by one of the two algorithms, it is shared with the other algorithm, thus increasing the other algorithm's chances to find even better solutions more quickly.

On the LKH side, the shared best solution is used as the initial solution of the next LKH iteration. This solution can potentially be better than the solution formed using LKH's construction heuristic. Recall that LKH is based on performing edge exchanges to improve a given initial solution.

On the B&B side, a better solution found by the LKH thread and shared with the B&B threads may enable pruning at shallower tree nodes. Pruning at a shallower node eliminates a larger subspace, thus speeding up the B&B search. Since LKH is not an exact algorithm, the combined algorithm is terminated when the parallel B&B algorithm completes searching the enumeration tree.

Finding quality solutions early speeds up a B&B algorithm, as it enables pruning at shallower tree nodes. In a pure B&B algorithm, finding quality solutions early requires optimizing the search order. However, optimizing the search order is one of the greatest challenges in designing a B&B algorithm, and parallelization makes this even more complicated [16]. In a parallel B&B algorithm, the difficulty of finding a good search order increases with the size and the complexity of the instance at hand. A parallel algorithm involves significant overhead, especially the overhead of synchronizing access to shared variables by multiple threads. This overhead increases with the number of threads. Therefore, running a strong metaheuristic in parallel with a B&B algorithm is a more effective way of finding quality solutions early than trying to optimize the search order of the parallel B&B algorithm.

By running the metaheuristic algorithm in a separate thread, we better utilize the available computational resources. The simple best-solution sharing mechanism that we use ensures that the synchronization overhead is minimized and that the algorithm scales up well as the number of threads is increased. Additionally, using a separate metaheuristic thread with best-solution sharing simplifies the integration process and helps reduce the complexity and avoid the potential pitfalls associated with an intricate parallel algorithms.

In summary, augmenting the B&B algorithm with a separate metaheuristic algorithm, along with an optimized mechanism for sharing the best solution, offers a practical and efficient strategy for enhancing the performance of a parallel B&B algorithm. This approach capitalizes on the advantages of parallel computing while mitigating the challenges associated with optimizing the search order.

5 EXPERIMENTAL RESULTS

5.1 Experimental Setup

The proposed parallel algorithm was tested on both the TSPLIB [31] and the SOPLIB [29] benchmark suites. The tests were run on a 32-core AMD Threadripper 2990WX processor with 128GB of memory. The operating system is Ubuntu 20.04. Thirty two threads were used in all experiments.

As in previous work [16], the TSPLIB and SOPLIB instances were classified into *easy*, *medium*, and *hard* based on the sequential algorithm's running time. An easy instance is an instance that the sequential algorithm solves within 10 seconds. A medium-difficulty instance is an instance that the sequential algorithm solves in more than 10 seconds but in less than an hour. An instance that the sequential algorithm cannot solve within an hour is classified as a hard instance. Easy instances are not considered in this paper. The number of instances in each difficulty category is shown in Table 1.

The proposed algorithm, as well as the pure B&B algorithm [16] and the LKH algorithm were applied to the medium and hard instances in SOPLIB and TSPLIB. The results are shown in the next subsections.

Table 1: Instance classification

Benchmark Suite	Total Instances	Easy Instances	Medium Instances	Hard Instances
SOPLIB	48	13	21	14
TSPLIB	41	12	7	22

5.2 Medium-Difficulty Instances

In Table 2, the proposed algorithm is compared to each of the pure B&B algorithm and the LKH algorithm on the medium-difficulty instances in SOPLIB (the upper sub-table) and TSPLIB (the lower sub-table). On SOPLIB, the proposed algorithm runs faster than the pure B&B algorithm on 8 instances and at the same speed on 13 instances. On those 13 instances, the solution times are not exactly the same, but they are within random variation. Running a parallel algorithm multiple times on the same instance does not produce exactly the same result each time. The results reported here are based on repeating each test three times and taking the median of the three measurements. Random variation is discussed in greater detail in our previous work [16].

In aggregate, the geometric-mean speedup of the proposed algorithm relative to the pure B&B algorithm on medium SOPLIB instances is 1.19, i.e., the proposed algorithm runs 19% faster, on average, on these instances. The maximum speedup on any instance is 2.79.

The results in Table 2 show that the proposed algorithm runs faster than LKH on 16 SOPLIB instances and slower than LKH on 5 SOPLIB instances. LKH tends to perform well on smaller instances and less-constrained instances. In this comparison, it is important to take into account that the proposed algorithm is an exact algorithm while LKH is a heuristic algorithm. An exact algorithm does not only find an optimal solution, but it also proves the optimality of that solution. So, it may spend a significant amount of time proving optimality after finding an optimal solution.

The geometric-mean speedup of the proposed algorithm relative to LKH on the medium SOPLIB instances is 10.17. The maximum speedup on any instance is 356. However, there is an instance on which the proposed algorithm runs 4 times slower than LKH.

Now, we discuss the comparison of the three algorithms on the medium-difficulty TSPLIB instances. The proposed algorithm runs faster than the pure B&B algorithm on 4 instances. In geometric mean, the proposed algorithm is 1.39 faster than pure B&B. In the best case, it runs 3.60 faster on one instance. In the worst-case, the proposed algorithm runs slightly slower on one instance. The proposed algorithm may run slower than the pure B&B algorithm, because the loss of resources from assigning a thread and a hardware core to the LKH algorithm may over-weigh the benefit from the better solutions found by LKH. The results in Table 2 show that this is unlikely to happen. On the vast majority of the instances, the proposed algorithm runs significantly faster than the pure B&B algorithm.

The results show that LKH runs much faster that the proposed algorithm on the medium TSPLIB instances. We noticed that LKH runs very fast on smaller TSPLIB instances with light precedence constraints.

Table 2: Comparison on medium-difficulty instances

SOPLIB	relative to pure B&B	relative to LKH
Faster instances	8	16
Equal-speed instances	13	0
Slower instances	0	5
Geo-mean speedup	1.19	10.17
Max speedup	2.79	356.18
Min speedup	1.00	0.24
TSPLIB	relative to pure B&B	relative to LKH
Faster instances	4	2
Equal-speed instances	3	0
Slower instances	0	5
Geo-mean speedup	1.39	0.08
Max speedup	3.60	4.36
Min speedup	0.94	0.0001

5.3 Hard Instances

In this subsection, we compare the proposed algorithm with the the pure B&B algorithm and LKH on the hard instances. With a time limit of 1 hour, the pure B&B algorithm finds the optimal solutions to 8 out of 14 hard SOPLIB instances and 2 out of 22 hard TSPLIB instances. The proposed algorithm optimally solves one more SOPLIB instance within an hour. Since both the proposed algorithm and the pure B&B algorithm time out on some hard instances, the comparison in this section focuses on the quality of the best solution (the final cost) computed by each algorithm. The results are shown in Table 3.

On the hard SOPLIB instances, the proposed algorithm finds better-cost solutions for 4 instances relative to pure B&B. Recall that the pure B&B algorithm solves 8 hard instances optimally, and thus the proposed algorithm cannot produce a better cost for these 8 instances. The proposed algorithm does not produce a worse cost for any instance. The geometric-mean improvement in cost relative to pure B&B is 4.49%. The maximum cost improvement on any instance is 22.11%.

The proposed algorithm finds better-cost solutions for all 14 hard SOPLIB instances relative to LKH. The geometric-mean improvement in cost is 33.14%, and the maximum cost improvement on any instance is 90.48%. These results show that the proposed algorithm takes advantage of the shared good solution produced by LKH and uses these solution to produce significantly better solutions.

It is noted that on the hard SOPLIB instances, the improvement of the proposed algorithm relative to LKH is much greater than the improvement relative to pure B&B, while on the hard TSPLIB instances, the opposite is true. This is attributed to the fact that the pure B&B algorithm performs much better than LKH on SOPLIB and much worse than LKH on TSPLIB. The combined algorithm successfully captures the strengths of both algorithms and produces significantly better overall results relative to any of them used individually.

Table 3: Comparison on hard instances

SOPLIB	relative to pure B&B	relative to LKH
Instances with lower cost	4	14
Instances with equal cost	10	0
Instances with higher cost	0	0
Geo-mean cost improvement	4.49%	33.14%
Max cost improvement	22.11%	90.48%
Min cost improvement	0.00%	5.49%
Extra instances solved optimally	1	9
TSPLIB	relative to pure B&B	relative to LKH
Instances with lower cost	20	4
Instances with equal cost	2	18
Instances with higher cost	0	0
Geo-mean cost improvement	6.91%	0.16%
Max cost improvement	21.75%	2.46%
Min cost improvement	0.00%	0.00%
Extra instances solved optimally	0	0

6 CONCLUSIONS AND FUTURE WORK

In this paper, we propose a hybrid algorithm that combines a parallel B&B algorithm and the LKH heuristic to solve the SOP. Our experimental evaluation using TSPLIB and SOPLIB shows that the combined algorithm greatly improves both the execution time and the quality of the solutions (the cost of the final solution) relative to each individual algorithm.

In future work, we will continue to work on enhancing the proposed algorithm to better utilize the shared best solution. In particular, we will focus on improving the proposed algorithm's ability to prove optimality if an optimal solution is found by either B&B or LKH.

ACKNOWLEDGMENTS

This work was supported in part by the US National Science Foundation (NSF) Award No 1911235.

REFERENCES

[1] D. Anghinolfi, R. Montemanni, M. Paolucci, and L.M. Gambardella. 2011. A hybrid particle swarm optimization approach for the sequential ordering problem. *Computers & Operations Research* 38, 7 (2011), 1076–1085.

[2] N. Ascheuer, L.F. Escudero, M. Grötschel, and M. Stoer. 1993. A cutting plane approach to the sequential ordering problem (with applications to job scheduling in manufacturing). *SIAM J. Optim.* 3, 1 (1993), 25–42.

[3] N. Ascheuer, M. Jünger, and G. Reinelt. 2000. A branch & cut algorithm for the asymmetric traveling salesman problem with precedence constraints. *Comput. Optim. Appl.* 17 (2000), 61–84.

[4] A. Borisenko and S. Gorlatch. 2019. Optimal batch plants design on parallel systems: A comparative study. In *IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*. 549–558.

[5] I. Chakroun and N. Melab. 2015. Towards a heterogeneous and adaptive parallel branch-and-bound algorithm. *J. Comput. System Sci.* 81, 1 (2015), 72–84.

[6] T.G. Crainic, B. Le Cun, and C. Roucairol. 2006. Parallel branch-and-bound algorithms. In *Parallel Combinatorial Optimization*. 1–28.

[7] A. Dabah, A. Bendjoudi, A. AitZai, D. El-Baz, and N.N. Taboudjemat. 2018. Hybrid multi-core CPU and GPU-based B&B approaches for the blocking job shop scheduling problem. *J. Parallel and Distrib. Comput.* 117 (2018), 73–86.

[8] A. de Bruin, G.A.P. Kindervater, and H.W.J.M. Trienekens. 1995. Asynchronous parallel branch and bound and anomalies. In *Parallel Algorithms for Irregularly Structured Problems*. Lecture Notes in Computer Science, Vol. 980. 363–377.

[9] L.F. Escudero. 1988. An inexact algorithm for the sequential ordering problem. *Eur. J. Oper. Res.* 37, 2 (1988), 236–249.

[10] L.F. Escudero, M. Guignard, and K. Malik. 1994. A Lagrangian relax-and-cut approach for the sequential ordering problem with precedence relationships. *Ann. Oper. Res.* 50, 1 (1994), 219–237.

[11] L.M. Gambardella, R. Montemanni, and D. Weyland. 2012. An enhanced ant colony system for the sequential ordering problem. In *Operations Research Proceedings 2011*. 355–360.

[12] B. Gendron and T.G. Crainic. 1994. Parallel branch-And-bound algorithms: Survey and synthesis. *Oper. Res.* 42, 6 (1994), 1042–1066.

[13] J. Gmys. 2017. *Heterogeneous cluster computing for many-task exact optimization - Application to permutation problems*. Ph. D. Dissertation. Université de Mons (UMONS) ; Université de Lille.

[14] J. Gmys, M. Mezmaz, N. Melab, and D. Tuyttens. 2020. A computationally efficient branch-and-bound algorithm for the permutation flow-shop scheduling problem. *European Journal of Operational Research* 284, 3 (2020), 814–833.

[15] T. Gonggiatgul, G. Shobaki, and P. Muyan-Özçelik. 2022. A Parallel Branch-and-Bound Algorithm with History-Based Domination. In *Proceedings of the ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPoPP)*. 439–440.

[16] T. Gonggiatgul, G. Shobaki, and P. Muyan-Özçelik. 2023. A parallel branch-and-bound algorithm with history-based domination and its application to the sequential ordering problem. *J. Parallel and Distrib. Comput.* 172 (2023), 131–143.

[17] L. Gouveia and P. Pesneau. 2006. On extended formulations for the precedence constrained asymmetric traveling salesman problem. *Networks* 48, 2 (2006), 77–89.

[18] L. Gouveia and M. Ruthmair. 2015. Load-dependent and precedence-based models for pickup and delivery problems. *Comput. Oper. Res.* 63 (2015), 56–71.

[19] K. Helsgaun. 2000. An effective implementation of the Lin–Kernighan traveling salesman heuristic. *European Journal of Operational Research* 126, 1 (2000), 106–130.

[20] K. Helsgaun. 2009. General k-opt submoves for the Lin–Kernighan TSP heuristic. *Math. Prog. Comp.* 1 (2009), 119–163.

[21] K. Helsgaun. 2017. *An extension of the Lin-Kernighan-Helsgaun TSP Solver for constrained traveling salesman and vehicle routing problems*. Technical Report. Roskilde Universitet.

[22] J. Jamal, G. Shobaki, V. Papapanagiotou, L.M. Gambardella, and R. Montemanni. 2017. Solving the sequential ordering problem using branch and bound. In *IEEE Symposium Series on Computational Intelligence*. 1–9.

[23] J. Kinable, A.A. Ciré, and W.-J. van Hoeve. 2017. Hybrid optimization methods for time-dependent sequencing problems. *Eur. J. Oper. Res.* 259, 3 (2017), 887–897.

[24] M. E. Lalami and D. El-Baz. 2012. GPU implementation of the branch and bound method for knapsack problems. In *IEEE International Parallel and Distributed Processing Symposium Workshops PhD Forum*. 1769–1777.

[25] S. Lin and B. W. Kernighan. 1973. An effective heuristic algorithm for the Traveling-Salesman Problem. *Operations Research* 21, 2 (1973), 498–516.

[26] C. McCreesh and P. Prosser. 2015. The shape of the search tree for the maximum clique problem and the implications for parallel branch and bound. *ACM Trans. Parallel Comput.* 2, 1 (2015), 8:1–8:27.

[27] A. Mingozzi, L. Bianco, and S. Ricciardelli. 1997. Dynamic programming strategies for the traveling salesman problem with time window and precedence constraints. *Oper. Res.* 45, 3 (1997), 365–377.

[28] M. Mojana, R. Montemanni, G.D. Caro, and L.M. Gambardella. 2012. A branch and bound approach for the sequential ordering problem. In *Proceedings of the International Conference on Applied Operational Research*. Lecture Notes in Management Science, Vol. 4. 266–273.

[29] R. Montemanni, D.H. Smith, and L.M. Gambardella. 2008. A heuristic manipulation technique for the sequential ordering problem. *Computers & Operations Research* 35, 12 (2008), 3931–3944.

[30] V. Papapanagiotou, J. Jamal, R. Montemanni, G. Shobaki, and L.M. Gambardella. 2015. A comparison of two exact algorithms for the sequential ordering problem. In *IEEE conference on systems, process and control (ICSPC)*. 73–78.

[31] G. Reinelt. 1991. TSPLIB—A traveling salesman problem library. *INFORMS Journal on Computing* 3, 4 (1991), 376–384.

[32] Y. Salii. 2019. Revisiting dynamic programming for precedence-constrained traveling salesman problem and its time-dependent generalization. *Eur. J. Oper. Res.* 272, 1 (2019), 32–42.

[33] Y.V. Salii and A.S. Sheka. 2020. Improving dynamic programming for travelling salesman with precedence constraints: Parallel Morin–Marsten bounding. *Optimization Methods and Software* (2020), 1–27.

[34] G. Shobaki and J. Jamal. 2015. An exact algorithm for the sequential ordering problem and its application to switching energy minimization in compilers. *Comput. Optim. Appl.* 61, 2 (2015), 343–372.

[35] R. Skinderowicz. 2017. An improved ant colony system for the sequential ordering problem. *Computers & Operations Research* 86 (2017), 1–17.