Tree searches for the Sequential Ordering Problem

Luc Libralesso and **Abdel-Malik Bouhassoun** and **Hadrien Cambazard** and **Vincent Jost**[1]

Abstract. We study several generic tree search techniques applied to the *Sequential Ordering Problem*. This study enables us to propose a simple yet competitive tree search. It consists of an iterative beam search that favors search over inference and integrates prunings that are inspired by dynamic programming. The resulting method proves optimality on half of the SOPLIB instances, 10 to 100 times faster than other existing methods. Furthermore, it finds new best-known solutions on 6 among 7 open instances of the benchmark in a small amount of time. These results highlight that there is a category of problems (containing at least SOP) where an anytime tree search is extremely efficient (compared to classical meta-heuristics) but was underestimated.

1 INTRODUCTION

While facing a hard operations research problem, two types of approaches are usually considered:

- Exact methods that find the optimal solutions at the price of a potentially very long computation time. Mixed Integer Programming, Constraint Programming are part of this category. This type of methods generally uses tree search techniques combined with strong bounds (cutting planes, branch-and-price, *etc.*)
- Meta-heuristics that allow to find near-optimal solutions. These methods are usually local-search-based (tabu-search, simulated annealing) or population-based (evolutionary algorithms).

It is common to find the following discourse in many publications, for instance [17]:

"*Tree search approaches like branch-and-bound are in essence designed to prove optimality [...] Moreover, tree search has an exponential behavior which makes it not scalable faced with real-world combinatorial problems inducing millions of binary decisions.*"

Classifying tree searches as exact methods is surprising since they are successfully used as heuristics, mainly in AI/planning [35, 26], and also sometimes in operations research [33, 29]. In this paper, we present an example in which tree searches can compete with classical meta-heuristics. We consider a well-studied operations research problem with a published benchmark (the SOP and the SOPLIB) where a large variety of intricate and advanced approaches have been applied for more than 30 years [11]. We show that a simple anytime tree search algorithm is competitive against classical meta-heuristics.

Actually, the resulting algorithm performed beyond expectations. It finds better solutions than the currently best-known ones on 6 among 7 open instances of the SOPLIB in a short amount of time (less than 600 seconds on a laptop computer compared to days for some other methods). Furthermore, this algorithm reports optimality proofs on large (very constrained) instances about 10 to 100 times faster than the existing exact approaches. This method is so simple that it could be considered as a baseline algorithm in AI/planning. Indeed, it consists of an iterative beam search with a closed-list mechanism using no heuristic information other than the prefix cost (*i.e.* $h(n) = 0$). This study shows that anytime tree searches are a crucial component (at least on the SOPLIB) and might deserve a greater consideration while designing optimization algorithms.

The source code and solutions can be downloaded at `https://gitlab.com/librallu/cats-ts-sop`.

This paper is structured as follows: Section 1 presents the Sequential Ordering Problem and a quick survey of existing methods. Section 2 presents the SOP specific bounds we use and compare (namely prefix bound, ingoing/outgoing bound, MST bound). Section 3 presents the generic branch and bounds parts we use (namely DFS, LDS, Beam Search and Prefix Equivalence). Finally, Section 4 presents numerical results on the impact of the search strategy and a comparison with existing state of the art algorithms.

1.1 SOP formal definition

Sequential Ordering Problem (SOP) is an Asymmetrical Traveling Salesman Problem with precedence constraints. See Figure 1 for an illustrative example.

An instance of SOP consists of a directed graph $G = (V, A)$, arc weights $w : A \to \mathbb{R}$, a set of precedence constraints $C \subseteq V \times V$ modeled as another graph, a start vertex $s \in V$, and a destination vertex $t \in V$. G is complete except for edges (u, v) where $(v, u) \in C$.

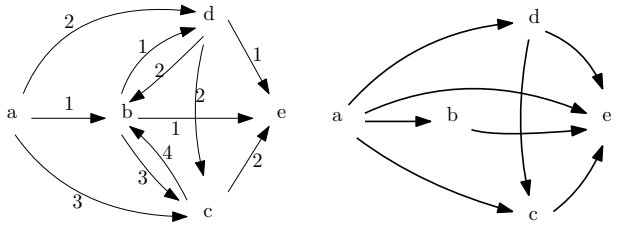
We search for a permutation of vertices that starts with s, ends with t, satisfies the precedence constraints (*i.e.* for each precedence constraint $(a, b) \in C$, vertex a must be visited before vertex b) and that minimizes the weighted sum of the arcs joining the vertices in the permutation.

1.2 Literature review

SOP was originally presented alongside some exact algorithms based on a mathematical programming model [11]. It has been extensively studied in the past 30 years, and many applications and resolution methods have been considered. SOP generalizes several combinatorial problems: Relaxing the precedence constraints gives the Asymmetric Traveling Salesman Problem (ATSP) [23]. If, moreover, arc lengths are symmetric, we get the symmetric TSP. We present in this section the most common applications and algorithms for SOP.

SOP arises in many industrial applications. On stacker crane trajectory optimization [2], one has to fulfill transportation jobs as fast as possible. This problem can be modeled using SOP where vertices represent jobs and arc weights represent the time needed to go from

[1] Univ. Grenoble Alpes, CNRS, Grenoble INP, G-SCOP, 38000 Grenoble, France, email: luc.libralesso@grenoble-inp.fr



(a) graph representation arc weights

(b) graph representation precedence constraints

Figure 1: Example of a SOP instance with 5 vertices and 1 precedence constraint where a is the start vertex and e the end vertex. Permutation a, d, c, b, e is a feasible (since d is visited before c) and has cost $2 + 2 + 4 + 2 = 10$. Permutations a, b, c, d, e and a, c, b, d, e are not feasible. Permutation a, b, d, c, e is optimal with cost $1+1+2+2 = 6$.

a job to another. In automotive paint shops [32] where the goal is to minimize the set-up cost of a paint job (flushing old paint, retrieving new color *etc.*). Also, since car lanes relative order cannot be changed during retrieval, precedence constraints need to be taken into account. SOP also occurs in the switching energy minimization of compilers [30]. While compiling a program, the compiler has to visit operations so that the switching cost is minimized. Since some operations require other operations to be done before starting, precedence constraints also need to be considered. One can also note the use of SOP in freight transportation [12], flexible manufacturing systems [2], and helicopter visiting [13].

Many exact approaches have been proposed to solve the Sequential Ordering Problem. As we discuss in this section, most of the literature focuses on finding strong lower bounds. Earlier approaches to SOP include cutting planes [3] and Lagrangian relax and cut algorithm [12]. A mathematical programming model solved with a branch and bound in which the branching is performed in order to decompose the problem as much as possible was also studied [24]. The uncapacitated m-PDTSP, which is a generalization of SOP, led to competitive results on SOP using a branch and cut algorithm combined with a generalized variable neighborhood search [18]. Also, decision diagrams made a huge impact by generating automatically good quality bounds [7, 21]. In 2015, a dedicated branch and bound has been proposed [30], it combines quick and elementary bounds (prefix, ingoing/outgoing degrees and MST) with a technique inspired from TSP dynamic programming called *History Cuts* that allows to prune dominated partial solutions. Despite the simplicity of its bounds, the later method obtained excellent numerical results. It therefore inspired us to study further the impact of the branch and bound components. This algorithm has been further improved by the integration of a custom assignment bound and a local-search at each node of the search tree [22].

In meta-heuristics, numerous works focus on a local-search move called SOP-3-exchange and combine it with various searches. It is a 3-OPT move optimized to take into account precedence constraints and asymmetrical arc weights. This SOP-3-exchange procedure is presented in alongside an Ant Colony Optimization algorithm [15]. It has also been used within a particle swarm optimization algorithm [1], by a hybrid genetic algorithm using a new crossover operator referred to as *Voronoi Quantized Crossover* [28], as well as a bee colony optimization [34], and a parallel roll-out algorithm [19].

Since the hybrid ant colony algorithm HAS-SOP [15] obtained excellent numerical results, a considerable amount of work has been done to improve it. First, by the integration of a better data structure called the *don't push stack* [16]. HAS-SOP was again improved by the integration of a Simulated Annealing scheme [31]. Recently, the LKH heuristic was improved to be able to solve SOP instances [20]. These two last methods obtained the best solutions on large instances of the SOPLIB.

According to the literature review on the Sequential Ordering Problem, the existing works seem to consider as a working hypothesis that local-search is a key feature to obtain state of the art solutions on large instances and that strong lower bounds are the key components of Branch and Bound algorithms. Moreover, since 2006, every work based on meta-heuristics use the SOPLIB as a standard benchmark, as we do here [1, 7, 16, 18, 19, 20, 22, 24, 30, 31, 34]. In the next sections of this paper, we investigate different branch and bound components and show that specific combinations can build very efficient methods that provide new best known solutions on large and constrained SOPLIB instances and prove optimality on half of them (which is not possible with most local-search strategies).

2 A SEARCH TREE FOR SOP

When designing a tree search algorithm, it is common to divide it into two parts. The search tree (problem specific part, *i.e.* how to branch, bounds, pruning, *etc.*) and the generic parts (a search strategy, such as DFS, Beam Search *etc.* or generic prunings, in our case domination prunings). This section presents the problem specific parts and the next section presents the generic parts.

During the implementation of the search trees, we focused on fast bounds ($O(1)$ for ingoing/outgoing bounds and $O(|E|)$ for the Minimum Spanning Tree bound). The key idea is to favor search over bounding/filtering. In the specific case of the SOPLIB, we show that using stronger bounds dramatically affects the performance of the method, even if the resulting branch and bound explores a smaller tree and has a better guidance.

We branch as follows: The root node contains the start vertex s. Each child of a given node corresponds to each possible next vertex to be visited (vertices not already added to the prefix and whose predecessors have all been added already to the prefix).

2.1 Definition and computation of lower bounds

We define our bounds as it is usually done in AI Planning. For a given node n we define the lower bound as follows: $f(n) = g(n) + h(n)$ where:

- $g(n)$ is the prefix bound (*i.e.* cost of arcs between already selected vertices)
- $h(n)$ is the suffix bound (*i.e.* an optimistic estimate of the remaining work to be done). The three bounds we develop in this section only differ on this criterion.

2.2 Prefix bound

The prefix bound consists in setting $f(n) = g(n)$ for any node of the search tree. That is $h(n) = 0$.

This bound (*i.e.* $g(n)$) can be computed in $O(1)$ along a branch of the search tree, simply by accessing, when adding vertex b to a prefix that ended with vertex a, the cost w_{ab} from the input. Within the scope of validity of our computational experiments, and despite its simplicity, this bound revealed itself as the best among the three bounds considered.

2.3 Ingoing/Outgoing bound

For the Ingoing/Outgoing (or I/O) bound, we keep the prefix bound and add a lower bound on the suffix.

Consider a node n of the search tree. Let $\text{prefix}(n) = v_1 \dots v_{k-1}$ be an ordered set of already visited vertices excluding the last added vertex v_k, and $\text{suffix}(n)$ the set of remaining vertices to add. We remind that s denotes the start vertex and t the end vertex of the SOP instance.

We design the optimistic estimate of remaining cost $h(n) = \max(h_{in}(n), h_{out}(n))$ where:

$$h_{in}(n) = \sum_{v \in \text{suffix(n)}} \min_{u \in V, uv \in A} w_{uv}$$

$$h_{out}(n) = \sum_{u \in (\text{suffix(n)} \cup \{v_k\}) \setminus \{t\}} \min_{v \in V, uv \in A} w_{uv}$$

This bound can be computed in $O(1)$ along a branch of the search tree. Indeed, one can precompute the sum of ingoing arcs at root node. When adding a vertex v to the prefix, this sum can be updated in constant time by removing the minimum ingoing arc for v. The same algorithm can be applied for outgoing arcs. Note that arcs considered in this bound can have one of their endpoint in the prefix. One can consider implementing a stronger bound that removes such arcs to further improve this bound.

2.4 Minimum spanning tree bound

For the MST bound, we keep the prefix bound and add a lower bound on the suffix.

Let $w_{ab} = +\infty$ if b must be visited before a. Define $w'_{ab} = \min(w_{ab}, w_{ba})$. The suffix cost $h(n)$ is then computed using Prim's algorithm on the graph spanned by the vertices not yet visited, with edge costs w'.

A key analysis, on the instances used for this paper, revealed that it would be pointless to try to speed-up the implementation of the MST bound, because, even if it could be computed as fast as the prefix bound, it would not lead to better solutions than the algorithms using this weaker bound. We ran an algorithm with MST within the time limit. Then ran algorithms with cheaper bounds restricting the number of nodes to the number opened by the MST based algorithm.

3 TREE SEARCH GENERIC COMPONENTS

In this paper, we examine several tree search components (namely the search strategy and the prefix equivalence, which can be seen as a form of dynamic programming or no-good recording embedded within a branch and bound scheme). We present a comprehensive study of the impact of these blocks and provide an efficient method based on this study.

3.1 Search strategies

Consider a search tree which is composed of a root node, bounds, and a children generation procedure for each node. The search strategy explores this tree, aiming to find the best possible solution, and, if possible, explores the whole tree which implies proving optimality. Since the 60s, new and efficient search strategies have been published. In this section, we describe some popular strategies within tree search algorithms that we use in our analysis.

3.1.1 Depth First Search

(DFS) explores a tree starting by the most promising child, explores the corresponding sub-tree entirely and eventually goes to the next child. This algorithm consumes a limited amount of memory while running ($O(nd)$ where n is the maximum number of children per node and d is the maximal depth of the tree).

However, DFS suffers from bad decisions made early in the tree exploration. Indeed, the search trees are usually so large that it is virtually impossible for DFS to overcome a bad decision taken at the root node. We note that many mechanisms such as random restarts or search strategies such as LDS have been designed to compensate this drawback of DFS. We precisely focus in this paper on such search strategies.

3.1.2 Limited Discrepancy Search

Given a maximum number of allowed discrepancies d, an iteration of LDS explores all nodes that have at most d deviations from the best child according to the guide (in our case the lower bound $f(n)$). Each node stores an allowed number of discrepancies. The root node starts with d allowed discrepancies. Its first best child is given d allowed discrepancies, its second best $d-1$ and so on. Nodes with negative discrepancies are not considered and pruned. This allows to explore the most promising branches of the tree while performing a restricted exploration of the other branches. It usually gives better solutions than DFS but can miss the optimal solutions. If $d = 1$, LDS behaves like a greedy algorithm. If $d = \infty$, LDS behaves like DFS.

Algorithm 1 shows the pseudo-code of an iterative LDS algorithm. We start by 1 allowed discrepancy. When the search ends, we restart with 2 allowed discrepancies until the stopping criterion is met.

Algorithm 1: Iterative LDS algorithm

Input : $G = (V, A)$, precedence constraints
Output : permutation of V

```
1  d ← 1;
2  while stopping criterion not met do
3      root.d ← d;
4      Stack ← root;
5      while Stack ≠ ∅ do
6          n ← Stack.pop();
7          i ← 0;
8          for c ∈ sortedChildren(n) do
9              c.d ← n.d − i;
10             Stack.push(c);
11             if c.d = 0 then
12                 break;
13             end
14             i ← i + 1;
15         end
16     end
17     d ← d + 1;
18 end
19 Report best solution found;
```

3.1.3 Beam Search

In LDS, nodes are selected depending on a comparison with their siblings and not depending on their absolute quality. We now present

Beam Search (BS) that aims to explore a subset of a tree that only keeps the best nodes at a given level. Beam Search has been used successfully to solve many scheduling problems [25, 27]. Beam Search is a tree search algorithm that uses a parameter called the beam size (D). Beam Search behaves like a truncated *Breadth First Search (BFS)*. It only considers the best D nodes on a given level. The others are discarded. Usually, we use the bound of a node to choose the most promising nodes. It generalizes both a greedy algorithm (if $D = 1$) and a BFS (if $D = \infty$). We consider the iterative version of beam search that performs a series of beam with sizes doubling at each iteration. We stop the algorithm either if the time limit is reached or the search tree is depleted by the last beam executed (in this case, we have proven the optimality).

Algorithm 2 shows the pseudo-code of an iterative beam search. The algorithm runs multiple beam searches starting with $D = 1$ (line 1) and increases geometrically the beam size (line 8). Each run explores the tree with the given parameter D. At the end of the time limit, we report the best solution found so far (line 10).

Algorithm 2: Iterative Beam Search algorithm

Input : $G = (V, A)$, precedence constraints
Output : permutation of V

```
1  D ← 1;
2  while stopping criterion not met do
3  |   Candidates ← {root} ;
4  |   while Candidates ≠ ∅ do
5  |   |   nextLevel ← ⋃_{n∈Candidates} children(n);
6  |   |   Candidates ← best D nodes among nextLevel;
7  |   end
8  |   D ← D× 2;
9  end
10 Report best solution found;
```

3.2 Prefix equivalence pruning

Prefix equivalence pruning are a way to eliminate symmetries and dominated partial-solutions. It can be seen as a form of dynamic programming integrated within a tree search algorithm. It stores all explored sub-states. Each node compares its prefix subset and last vertex to existing entries in the database. If it is dominated, the node is pruned. This strategy has been used in a large variety of methods. For instance, memorization in branch and bounds ([30, 29]).

A prefix equivalence pruning for the Sequential Ordering Problem can be defined as follows (inspired from TSP dynamic programming [6], history cuts [30] and the call-based dynamic programming [4]):

Two solution prefixes n_1, n_2 are called *equivalent* if they cover the same subset $S \subseteq V$ of vertices and end with the same last vertex v. If the prefix cost $g(n_1)$ (*i.e.* the sum of selected arcs between vertices from $S \cup \{v\}$) is (strictly) greater than $g(n_2)$, then n_1 is (strictly) dominated by n_2 and thus can be pruned.

In other words, the Prefix Equivalence prunings can be seen as a form of dynamic programming where the formulation can be described as follows where pred(S, j) indicates that j is not a predecessor of any vertex in S:

$$f^*(S, i) = \min_{j \in S \wedge \text{pred}(S,j)}(f^*(S \setminus \{j\}, j) + w_{ji}$$

Our implementation of Prefix equivalence consists in altering the behavior of the branch and bound as follows: Each time a node n is opened, the prefix of n is compared to what exists in the database. If the subset of vertices spanned by n does not exist in the database it is added to it, otherwise it is compared to the best equivalent prefix found so far. If the subset has a prefix cost worst than the one in the database, the node n is pruned.

We implement the database using a hash table. In our numeric experiments, we notice that a branch and bound using the prefix equivalence opens on average 4 to 5 times less nodes than its equivalent version without prefix equivalence.

In some versions of the Prefix Equivalence (for instance the one found in history cuts [30]), nodes are pruned if their prefix matches an existing entry in the database even if their cost is equal. Notice that we restart tree searches (*i.e.* Iterative Beam Search and Limited Discrepancy), which perform heuristic prunings (they prune nodes to avoid saturating the memory and to ensure reaching feasible solutions). To allow our algorithms to close an instance (*i.e.* to prove the optimality of the best solution it found), we prune nodes only if they are *strictly* dominated by the best equivalent recorded in the database. The reason for doing so is that, although the value recorded in the database corresponds to a node that has been already explored, this exploration might have been partial and we need to ensure that the search does not perform any heuristic pruning to provide a proof of optimality.

4 COMPUTATIONAL RESULTS

Results were obtained from a Intel(R) Core(TM) i5-3470 CPU @ 3.20GHz with 8GB RAM. We run each pair of instance-algorithm for 600 seconds. Instances come from the SOPLIB benchmark available here

`http://www.idsia.ch/~roberto/SOPLIB06.zip`

The Instances are randomly generated and their names contain 3 numbers indicating: the number of nodes (from 200 to 700), the range of the cost drawn uniformly (either between 0 and 100 or between 0 and 1000), and the percentage of precedence constraints. Notice that the Instance $R.200.100.60$ is ill defined as its costs are drawn between 0 and 1000.

Best known bounds and solutions are an aggregation of results coming from [31, 18, 24, 22, 20]. Note that the Lin Kernighan Helsgaun 3 Algorithm [20] was run on each instance for 100.000 seconds. The Enhanced Ant Colony System with Simulated Annealing [31] was run 30 times per instance for 600 seconds so 18.000 seconds per instance. The time limit of 600s used in the present paper is therefore considerably smaller.

Performance of tree search components We ran 18 different tree searches (DFS, LDS and Beam Search) with and without Prefix equivalence using the prefix, Ingoing/Outgoing degrees or the MST bound for 600 seconds. It turns out that there are two clear winners out of these methods (Beam Search + Prefix Equivalence + Prefix or Ingoing/Outgoing bound). Since the results of the two best methods are very similar, we choose to put the emphasis on the simplest one (*i.e.* Beam Search + Prefix Equivalence + Prefix bound). We show in Table 1 that any deviation of search strategy, prunings or bounds lead to a performance drop (except for the Beam Search + Prefix Equivalence + Ingoing/Outgoing degree bound). For the sake of clarity, we only show deviations of BS+PE+P and not the 18 algorithms.

Discussion As expected, tree-search performs better (even more with prefix-equivalence prunings) on most-constrained instances (proving optimality on the 60% and 30% and competitive results on the 15%) while obtaining poor results on loosely constrained ones (1%)

The MST based tree searches open less nodes (1.000 to 10.000 times less than the ingoing/outgoing bound). This leads to less solutions found (sometimes none within the time limit) and is overall less efficient. It appears that on medium size instances, the MST bound does not provide a significant guide improvement (and thus harms performance since it is more expensive to compute than the Prefix or Ingoing/Outgoing bound). One might wonder whether a possible incremental evaluation of the MST bound, that is, a computation of it along a branch of the search tree taking advantage of the similarity between the MST for a node and the MST for one of its child, would make a difference. For the benchmark we used, it would make absolutely no difference. In the best scenario, we would end up with a third algorithm equivalent to our other two champions. We do not report numerical results on this issue here, but restricting the algorithms by the number of nodes, and not by time limit, we observed that the MST bound did not improve the results overall.

We remark that the search strategy also plays an important role while finding good solutions or closing instances within the time limit. Globally, the Beam Search strategy finds better solutions than LDS which in turns finds better solutions than DFS. Although DFS is able to find the optimal solution and to prove optimality on some instances, it doesn't match the quality of the solutions of either Beam Search or LDS. The main advantage of DFS is that it does not reopen any node, its main drawback is that it struggles to provide good quality solutions fast. In comparison, Beam Search reopens nodes, but by finding very good solutions fast, it is able to prune more nodes and thus, close more instances. In this study, the beam search strategy (using prefix equivalence) appears to be the best strategy, both for proving optimality and finding the best solutions within the time limit.

We compare the *Beam Search + Prefix Equivalence + Prefix or Ingoing/Outgoing bound* against the best solutions reported in the literature by other state of the art algorithms. Our method finds new best known solutions on 6 among 7 open instances of the SOPLIB in a much shorter time than the other algorithms. It also proves optimality quickly on all instances with 30 and 60 percent precedence (about 10 to 100 times faster than the DFS+prefix equivalence+stronger bounds+local-search [22]). We remark that the proposed method fails to provide good solutions for 1% precedence due to the poor quality of bounds on these instances that are close to ATSP.

5 CONCLUSION AND FUTURE WORKS

In this paper, we discussed the importance considering anytime tree searches while designing algorithms for large-scale instances. In this (striking) example, we showed that even a simple tree-search algorithm (that could also be considered as a baseline algorithm due to its simplicity) could outperform state-of-the-art operations research intricate and advanced meta-heuristics on a well studied benchmark.

The search algorithms usually considered in operations research (namely DFS and LDS) proved to be clearly dominated by the beam search performance-wise on the SOPLIB. Beam search outperformed DFS by proving optimality on 25 instances (DFS proved optimality only on 17 instances) in less than 600 seconds. It outperformed existing algorithms in finding new best known solutions on 6 among 7 open instances in a short amount of time. We also demonstrated the importance of deconstructing optimization algorithms (in this case a tree-search) and of the analysis of contribution and computational cost of each separate building block. Indeed, neither beam search alone, nor the prefix equivalence prunings with DFS/LDS alone were able to outperform state-of-the art, but together, they did.

While aiming to implement an efficient all-purpose SOP solver, one (in our opinion) should integrate a Beam Search and prefix equivalence as it proves itself to be a very efficient and complementary approach on large and constrained instances. However, we note that such tree-search methods are not as efficient on loosely constrained instances as LP-based branch-and-bounds or local-search, thus the importance of hybridizing them together in such all-purpose solver.

The existing SOPLIB contains mostly very constrained instances (at the exception of the 1% precedencies). It was probably designed in such a way that local-search an LP-based approaches struggle to find good solutions. As tree-search outperform these approaches on 15% instances and is outperformed on 1% instances, it is an interesting question to update the benchmark with intermediate densities (5%, 10% *etc.*) as they would probably be harder considering both classes of algorithms.

This paper only considers the Sequential Ordering Problem. However, a similar decomposition methodology of complicated algorithms into simple building blocks and the assessment of their contributions, computational costs, and ideally synergies, can be applied on other combinatorial optimization problems. For instance, anytime tree searches have been successfully applied on various hard combinatorial optimization problems such as "simple assembly line balancing problem" [5], "Longest Palindromic Common Sub-sequence" [10]. We believe that similar conclusions (anytime tree-search are underestimated by operations research practitioners) can be drawn on several other problems.

Moreover, we limited this study to DFS, LDS and Beam Search as search strategies. Many more exist (like Beam stack search [35], BULB [14], Anytime Focal Search [8] *etc.*). Also, one can study other branch and bound components like performing a local-search within each node [22, 9] or the probing strategy (starting a greedy algorithm at each node to obtain a better quality estimate). This would probably lead to better insights on when and how using anytime tree searches while facing operations research problems.

To try and evaluate the contribution of such building blocks and ideas on various problems, we started developing a framework for generic Tree-Search, which allows to define a combinatorial problem as a branching scheme, specific prunings, LP cuts, and bounds and then to apply generic Tree Search strategies. This might lead to a new standard way (aside Mathematical Programming, Constraint Programming, local-search [17], Satisfiability of Boolean Clauses, *etc.*) to define and solve combinatorial problems.

ACKNOWLEDGEMENTS

We would like to thank Louis Esperet for his useful comments. Also, we would like to thank the referees for their comments, which helped improve this paper considerably.

Instance	BKLB	BKUB	BS,PE,P	BS,PE,IO	BS,PE,MST	BS,P	DFS,PE,P	LDS,PE,P	T record (s)	T opt (s)
R.200.100.1	61	61	189	189	299	189	283	192	-	-
R.200.100.15	1.792	1.792	1.792	1.792	1.887	1.796	3.740	2.325	19.5	-
R.200.100.30	4.216	4.216	**4.216**	**4.216**	**4.216**	4.249	**4.216**	4.216	0.1	0.6
R.200.100.60	71.749	71.749	**71.749**	**71.749**	**71.749**	71.749	**71.749**	71.749	0.0	0.0
R.200.1000.1	1.404	1.404	2.554	2.554	3.398	2.554	3.448	2.684	-	-
R.200.1000.15	20.481	20.481	**20.481**	**20.481**	20.952	20.517	34.982	25.592	16.3	547.7
R.200.1000.30	41.196	41.196	**41.196**	**41.196**	**41.196**	41.728	**41.196**	41.196	0.1	0.4
R.200.1000.60	71.556	71.556	**71.556**	**71.556**	**71.556**	71.556	**71.556**	71.556	0.0	0.0
R.300.100.1	26	26	214	214	406	204	265	225	-	-
R.300.100.15	3.152	3.152	3.152	3.152	3.458	3.201	5.355	4.081	178.9	-
R.300.100.30	6.120	6.120	**6.120**	**6.120**	6.330	6.200	**6.120**	6.120	2.2	7.9
R.300.100.60	9.726	9.726	**9.726**	**9.726**	**9.726**	9.726	**9.726**	9.726	0.0	0.0
R.300.1000.1	1.294	1.294	3.080	3.080	4.784	2.888	3.551	3.012	-	-
R.300.1000.15	29.006	29.006	29.006	29.006	33.885	29.481	51.152	43.597	220.0	-
R.300.1000.30	54.147	54.147	**54.147**	**54.147**	54.491	54.533	55.791	54.147	0.4	3.6
R.300.1000.60	109.471	109.471	**109.471**	**109.471**	**109.471**	109.471	**109.471**	109.471	0.0	0.0
R.400.100.1	13	13	191	191	-	175	295	203	-	-
R.400.100.15	3.879	3.879	3.879	3.879	5.011	3.961	8.103	6.584	176.7	-
R.400.100.30	8.165	8.165	**8.165**	**8.165**	8.210	8.183	**8.165**	8.165	0.4	2.1
R.400.100.60	15.228	15.228	**15.228**	**15.228**	**15.228**	15.228	**15.228**	15.228	0.0	0.0
R.400.1000.1	1.343	1.343	3.247	3.247	-	3.247	4.466	3.525	-	-
R.400.1000.15	35.364	38.963	38.963	38.963	53.789	39.722	76.463	69.342	157.2	-
R.400.1000.30	85.128	85.128	**85.128**	**85.128**	87.698	85.720	**85.128**	85.128	0.5	1.8
R.400.1000.60	140.816	140.816	**140.816**	**140.816**	**140.816**	140.816	**140.816**	140.816	0.0	0.0
R.500.100.1	4	4	267	275	-	202	272	232	-	-
R.500.100.15	4.628	5.284	5.261	5.261	7.593	5.411	9.917	9.610	206.5	-
R.500.100.30	9.665	9.665	**9.665**	**9.665**	10.388	9.778	10.999	9.665	1.4	6.2
R.500.100.60	18.240	18.240	**18.240**	**18.240**	**18.240**	18.257	**18.240**	18.240	0.0	0.1
R.500.1000.1	1.316	1.316	4.079	4.079	-	3.541	4.703	3.717	-	-
R.500.1000.15	43.134	49.504	49.366	49.366	71.888	50.624	103.985	94.625	120.8	-
R.500.1000.30	98.987	98.987	**98.987**	**98.987**	115.074	99.492	114.544	98.987	1.7	3.8
R.500.1000.60	178.212	178.212	**178.212**	**178.212**	**178.212**	178.355	**178.212**	178.212	0.0	0.0
R.600.100.1	1	1	289	289	-	289	306	246	-	-
R.600.100.15	4.803	5.472	5.469	5.469	9.329	5.695	13.007	10.939	160.5	-
R.600.100.30	12.465	12.465	**12.465**	**12.465**	12.929	12.475	13.899	12.465	3.1	10.3
R.600.100.60	23.293	23.293	**23.293**	**23.293**	**23.293**	23.324	**23.293**	23.293	0.0	0.0
R.600.1000.1	1.337	1.337	4.030	4.030	-	3.853	4.814	4.074	-	-
R.600.1000.15	47.042	55.213	54.994	54.994	90.977	55.748	115.295	108.164	183.6	-
R.600.1000.30	126.789	126.789	**126.798**	**126.798**	158.425	128.761	145.672	126.798	1.6	7.2
R.600.1000.60	214.608	214.608	**214.608**	**214.608**	214.608	214.608	**214.608**	214.608	0.1	0.1
R.700.100.1	1	1	186	250	-	186	281	258	-	-
R.700.100.15	5.946	7.021	7.020	7.020	11.392	7.220	13.778	13.206	386.9	-
R.700.100.30	14.510	14.510	**14.510**	**14.510**	17.125	14.632	19.655	14.510	4.2	21.6
R.700.100.60	24.102	24.102	**24.102**	**24.102**	24.848	24.102	**24.102**	24.102	0.2	0.5
R.700.1000.1	1.231	1.231	4.403	4.403	-	4.042	4.629	4.028	-	-
R.700.1000.15	54.351	65.305	64.777	64.777	108.627	65.775	141.544	121.189	25.5	-
R.700.1000.30	134.474	134.474	**134.474**	**134.474**	158.327	136.073	158.613	134.474	1.3	4.8
R.700.1000.60	245.589	245.589	**245.589**	**245.589**	245.688	245.752	**245.589**	245.589	0.1	0.1
nb closed			25	25	11	0	17	0		

Table 1: Tree search components performance.

BKLB (resp. BKUB) refers to the Best Known Lower Bound (resp. Upper Bound) from our literature review.
BS,PE,P refers to the combination of Beam Search, Prefix Equivalence and Prefix bound.
BS,PE,IO refers to Beam Search, Prefix Equivalence and Ingoing/Outgoing bound.
BS,PE,MST refers to the Beam Search with Prefix Equivalence and MST bound.
BS,P refers to the Beam Search with the Prefix bound and without Prefix Equivalence.
DFS,PE,P refers to Depth First Search with Prefix Equivalence and Prefix bound.
LDS,PE,P refers to limited Discrepancy Search with Prefix Equivalence and Prefix bound.
"T record" indicates the time required by BS,PE,P to reach a solution of value BKUB or lower.
"T opt" indicates the time required by BS,PE,P to close the instance.

Bold instances are still open.
"-" (in column BS,PE,MST) indicate that no solution has been found by the method within 600 seconds.
"-" (in columns record (resp. opt)) indicate that no new record (resp. proof of optimality) was found by any of our methods.
Bold objective values indicate when the method was able to close the instance within the time limit.
Underlined objective values indicate an improvement of best known solutions.

REFERENCES

[1] Davide Anghinolfi, Roberto Montemanni, Massimo Paolucci, and Luca Maria Gambardella, 'A hybrid particle swarm optimization approach for the sequential ordering problem', *Computers & Operations Research*, **38**(7), 1076–1085, (2011).

[2] Norbert Ascheuer, 'Hamiltonian path problems in the on-line optimization of flexible manufacturing systems', *Technical Report TR 96–3*, (1996).

[3] Norbert Ascheuer, Laureano F Escudero, Martin Grötschel, and Mechthild Stoer, 'A cutting plane approach to the sequential ordering problem (with applications to job scheduling in manufacturing)', *SIAM Journal on Optimization*, **3**(1), 25–42, (1993).

[4] Thierry Benoist, Antoine Jeanjean, and Vincent Jost, 'Call-based dynamic programming for the precedence constrained line traveling salesman', in *International Conference on AI and OR Techniques in Constriant Programming for Combinatorial Optimization Problems*, pp. 1–14. Springer, (2014).

[5] Christian Blum, 'Beam-aco for simple assembly line balancing', *INFORMS Journal on Computing*, **20**(4), 618–627, (2008).

[6] Chetan Chauhan, Ravindra Gupta, and Kshitij Pathak, 'Survey of methods of solving tsp along with its implementation using dynamic programming approach', *International journal of computer applications*, **52**(4), (2012).

[7] Andre A Cire and Willem-Jan van Hoeve, 'Multivalued decision diagrams for sequencing problems', *Operations Research*, **61**(6), 1411–1428, (2013).

[8] Liron Cohen, Matias Greco, Hang Ma, Carlos Hernández, Ariel Felner, TK Satish Kumar, and Sven Koenig, 'Anytime focal search with applications.', in *IJCAI*, pp. 1434–1441, (2018).

[9] Federico Della Croce, Marco Ghirardi, and Roberto Tadei, 'Recovering beam search: Enhancing the beam search approach for combinatorial optimization problems', *Journal of Heuristics*, **10**(1), 89–104, (2004).

[10] Marko Djukanovic, Günther R Raidl, and Christian Blum, 'Anytime algorithms for the longest common palindromic subsequence problem', *Computers & Operations Research*, 104827, (2019).

[11] Laureano F Escudero, 'An inexact algorithm for the sequential ordering problem', *European Journal of Operational Research*, **37**(2), 236–249, (1988).

[12] Laureano F Escudero, Monique Guignard, and Kavindra Malik, 'A lagrangian relax-and-cut approach for the sequential ordering problem with precedence relationships', *Annals of Operations Research*, **50**(1), 219–237, (1994).

[13] Marie T Fiala Timlin and William R Pulleyblank, 'Precedence constrained routing and helicopter scheduling: heuristic design', *Interfaces*, **22**(3), 100–111, (1992).

[14] David Furcy and Sven Koenig, 'Limited discrepancy beam search', in *IJCAI*, pp. 125–131, (2005).

[15] Luca Maria Gambardella and Marco Dorigo, 'Has-sop: Hybrid ant system for the sequential ordering problem', *Technical Report IDSIA 11-97*, (1997).

[16] Luca Maria Gambardella, Roberto Montemanni, and Dennis Weyland, 'An enhanced ant colony system for the sequential ordering problem', in *Operations Research Proceedings 2011*, 355–360, Springer, (2012).

[17] Frédéric Gardi, Thierry Benoist, Julien Darlay, Bertrand Estellon, and Romain Megel, *Mathematical programming solver based on local search*, Wiley Online Library, 2014.

[18] Luis Gouveia and Mario Ruthmair, 'Load-dependent and precedence-based models for pickup and delivery problems', *Computers & Operations Research*, **63**, 56–71, (2015).

[19] Francesca Guerriero and Marco Mancini, 'A cooperative parallel rollout algorithm for the sequential ordering problem', *Parallel Computing*, **29**(5), 663–677, (2003).

[20] Keld Helsgaun, 'An extension of the lin-kernighan-helsgaun tsp solver for constrained traveling salesman and vehicle routing problems', *Roskilde: Roskilde University*, (2017).

[21] István T Hernádvölgyi, 'Solving the sequential ordering problem with automatically generated lower bounds', in *Operations Research Proceedings 2003*, 355–362, Springer, (2004).

[22] J Jamal, G Shobaki, Vassilis Papapanagiotou, Luca Maria Gambardella, and Roberto Montemanni, 'Solving the sequential ordering problem using branch and bound', in *2017 IEEE Symposium Series on Computational Intelligence (SSCI)*, pp. 1–9. IEEE, (2017).

[23] Gilbert Laporte and Ibrahim H Osman, 'Routing problems: A bibliography', *Annals of Operations Research*, **61**(1), 227–262, (1995).

[24] Marco Mojana, Roberto Montemanni, Gianni Di Caro, Luca M Gambardella, and P Luangpaiboon, 'A branch and bound approach for the sequential ordering problem', *Lecture Notes in Management Science*, **4**, 266–273, (2012).

[25] Peng Si Ow and Thomas E Morton, 'Filtered beam search in scheduling', *The International Journal Of Production Research*, **26**(1), 35–62, (1988).

[26] Stuart J Russell and Peter Norvig, *Artificial intelligence: a modern approach*, Malaysia; Pearson Education Limited,, 2016.

[27] Ihsan Sabuncuoglu and M Bayiz, 'Job shop scheduling with beam search', *European Journal of Operational Research*, **118**(2), 390–412, (1999).

[28] Dong-Il Seo and Byung-Ro Moon, 'A hybrid genetic algorithm based on complete graph representation for the sequential ordering problem', in *Genetic and Evolutionary Computation Conference*, pp. 669–680. Springer, (2003).

[29] Lei Shang, Vincent T'Kindt, and Federico Della Croce, 'The memorization paradigm: Branch & memorize algorithms for the efficient solution of sequencing problems', ., (2018).

[30] Ghassan Shobaki and Jafar Jamal, 'An exact algorithm for the sequential ordering problem and its application to switching energy minimization in compilers', *Computational Optimization and Applications*, **61**(2), 343–372, (2015).

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

[32] Sven Spieckermann, Kai Gutenschwager, and Stefan Voß, 'A sequential ordering problem in automotive paint shops', *International journal of production research*, **42**(9), 1865–1878, (2004).

[33] Vincent T'kindt, Federico Della Croce, and Carl Esswein, 'Revisiting branch and bound search strategies for machine scheduling problems', *Journal of Scheduling*, **7**(6), 429–440, (2004).

[34] Moon Hong Wun, Li-Pei WongT, Ahamad Tajudin Khader, and Tien-Ping Tan, 'A bee colony optimization with automated parameter tuning for sequential ordering problem', in *2014 4th World Congress on Information and Communication Technologies (WICT 2014)*, pp. 314–319. IEEE, (2014).

[35] Rong Zhou and Eric A Hansen, 'Beam-stack search: Integrating backtracking with beam search.', in *ICAPS*, pp. 90–98, (2005).