Branch-and-bound for the Precedence Constrained Generalized Traveling Salesman Problem

Raad Salman[1*] salman@fcc.chalmers.se
Fredrik Ekstedt[1] fredrik.ekstedt@fcc.chalmers.se
Peter Damaschke[2] ptr@chalmers.se

[*] Corresponding author.
[1] Fraunhofer-Chalmers Centre, Chalmers Science Park, 412 88 Gothenburg, Sweden
[2] Department of Computer Science and Engineering, Chalmers University of Technology, 412 96 Gothenburg, Sweden

October 18, 2017

Abstract

The Precedence Constrained Generalized Traveling Salesman Problem (PCGTSP) asks to find a cheapest closed tour through groups of vertices, visiting one vertex from each group and respecting precendence constraints between certain pairs of groups. While the Generalized TSP (GTSP) and the precedence constrained TSP are well-known problems, powerful optimization methods for the combination of both problems are lacking so far. This paper presents a branch-and-bound method for the PCGTSP. Different ways to bound the subproblems in the search tree by transformations and relaxations are evaluated. Our algorithm incorporates shortest-path calculations and utilizes history from the search tree evaluation to limit branching. We also introduce an apparently new way of using the assignment problem to get lower bounds for the GTSP. Results show that the algorithm solves problem instances with 12-26 groups within a minute, and instances with around 50 groups which are more dense with precedence constraints within 24 hours on a PC with an Intel i7-6700k CPU and 32GB of RAM.

Keywords: generalized traveling salesman problem; precedence constraints; sequential ordering problem; branch-and-bound; assignment problem; minimum spanning arborescence problem

1 Introduction

In this paper we consider the precedence constrained generalized traveling salesman problem (PCGTSP). This variant of the asymmetric traveling salesman problem (ATSP) is defined on a directed, edge-weighted graph where the set of all vertices is partitioned into a number of disjoint sets called *groups*. Furthermore, pairwise relationships between groups, called *precedence constraints*, dictate that for some groups others must precede them in any feasible solution. The PCGTSP asks to find a minimum-cost closed tour (starting at a specified start group and eventually returning) such that exactly one vertex in each group is visited in a valid order with respect to the precedence constraints. Informally, we seek the cheapest Hamiltonian tour through the groups that also respects the precedence constraints. Problems of this type can arise, for example, in industrial processes where tasks which can be performed in many different ways are to be sequenced with respect to some order which ensures the integrity of the process. Optimizing such sequences of tasks is of great importance when striving for sustainable and efficient manufacturing.

1.1 Related Literature

PCGTSP is closely related to the sequential ordering problem (SOP) and the generalized TSP (GTSP), both of which have been studied extensively. In [13] Escudero et al. present a Langrangian relaxation scheme for obtaining lower bounds for the SOP. To incorporate the precedence constraints in the bounds, a set of cuts are generated when solving each Lagrangian subproblem. Ascheuer et al. [4] develop a

branch-and-cut algorithm for the SOP with many new valid inequalities based on the work of Balas et al. in [6] and Ascheuer in [2]. Balas [5] shows that the precedence constrained ATSP (which is equivalent to the SOP) is solvable by a dynamic programming algorithm (first presented for the TSP by Held & Karp in [20]) in linear time if certain conditions on the precedence constraints are met. Hernádvölgyi [22] and Cire & Hoeve [9] obtain bounds on the SOP by restricting the state space of the dynamic programming. Gouveia & Ruthmair develop a branch-and-cut with many strong cuts for the SOP which closed many open benchmark instances. Montemanni et al. [26] develop a decomposition based branch-and-bound algorithm which attempts to solve the SOP by dividing an instance into smaller problems based on the parital order imposed by the precedence constraints. Shobaki & Jamal [32] evaluate a branch-and-bound algorithm with a simpler bounding algorithm but use a powerful pruning technique where information from previous tree nodes is reused. Recently the branch-and-cut algorithm developed in [17] closed many open benchmark instances for the SOP. Many different types of metaheuristic approaches such as genetic algorithms, particle swarm optimization and ant colony optimization have been presented in [1, 16, 35] amongst others. These are often fast but cannot guarantee optimality.

Fischetti et al. [15] present a branch-and-cut algorithm for the symmetric GTSP. They derive many valid inequalities based on the investigation of the GTSP polytope in [14]. The asymmetric case of the GTSP was considered by Laporte et al. in [24] where an integer programming formulation and branch-and-bound algorithm is presented. Noon & Bean [28] present a Lagrangian dual based branch-and-bound algorithm for the asymmetric GTSP. Bounds were obtained by solving either an assignment problem or a TSP of size equal to the number of groups to optimality. Different types of heuristics valid for both the symmetric and asymmetric GTSP have been developed and evaluated in [18, 21, 23, 33, 34].

The PCGTSP has recently attracted interest but still remains a largely unexplored problem. Chentsov et al. [8] extend the work done by Balas in [5] and present a dynamic programming algorithm which is able to solve PCGTSP instances within polynomial time under certain conditions. Dewil et al. in [11] and Dewil et al. in [10] present heuristics for constructing and improving solutions for the laser cutting tool problem which they model as a PCGTSP with some additional constraints. In [7] Castelino et al. use a transformation to go from a problem with a partitioned vertex set to a problem which is unpartitioned. The resulting SOP is then solved using heuristics developed by Ascheuer et al. in [3]. Salman et al. [30] present heuristics for the PCGTSP with a focus on industrial applications and the surrounding process which generates the problem.

1.2 Contribution and Outline

The main contributions of this paper are:

- Presenting a first exact branch-and-bound based algorithm for solving the PCGTSP.
- A new branching strategy where feasible sequences of groups are enumerated and shortest-path calculations are utilized in order to formulate the subproblems.
- A comparison of different methods for bounding the subproblems.
- A generalization of the history utilization pruning method developed in [32].
- A novel way of computing an assignment problem based bound for the GTSP.

Section 2 describes the PCGTSP formally and introduces the notation used in the paper. In Section 3 the branch-and-bound algorithm, the different bounding methods and the history utilization technique are presented. In Section 4 the algorithm is tested on industrial and synthetic instances, and in Section 5 the results of the tests are discussed and some suggestions for future development are given.

2 Problem Description

A PCGTSP instance $\mathcal{P}$ is defined by a directed graph $G = (V, A)$ where $V := \{1, \ldots, n\}$ denotes the set of vertices, $A := \{(i, j) : i, j \in V, i \neq j\}$ denotes the set of arcs, and a cost c_{ij} is associated with each arc $(i, j) \in A$. Additionally we let $\{V_1, \ldots, V_m\}$ be a partition of V where V_p, $p \in M$, is called a *group*, and let $M := \{1, \ldots, m\}$. For each vertex $i \in V$ we define $g(i)$ to be the index of the group which contains vertex i, that is $i \in V_{g(i)}$. We let the precedence constraints be defined by an acyclic digraph $G' = (M, \Pi)$ so that if $(p, q) \in \Pi$ then group p must precede group q in a feasible tour. Π includes all

relationships implied by the precedence constraints by transitivity. That is, if $(p, q) \in \Pi$ and $(q, r) \in \Pi$ then $(p, r) \in \Pi$.

The PCGTSP then asks to find a minimum-cost closed tour such that exactly one vertex in each group is visited and the precedence constraints are fulfilled. The tour must start and end in group V_1, therefore $(1, q) \in \Pi$ for all $q \in M \setminus \{1\}$. Note that in any feasible tour the precedence constraints apply only to the path without the last arc returning to V_1, whereas the cost of this last arc is included in the sum of arc costs.

We use the integer variable v_k to denote the group index sequenced at position k in the tour. Note that for any feasible solution we will have that $v_1 = 1$ since the group V_1 is assumed to be the start group.

3 Overview of Methods

We propose a branch-and-bound enumeration procedure where the PCGTSP is appropriately transformed and relaxed in order to obtain lower bounds. Below, we will discuss branching strategies, the various methods used for computing lower bounds and other methods used to limit branching. We will use $C_{\min}(\cdot)$ as a generic notation for the cost of the optimal solution for a problem instance.

3.1 Branching

We have opted to branch on the group sequence variables v_i which essentially means that each branch corresponds to a choice of which group to visit next in the tour. While branching on Boolean edge variables is a natural and common approach in an integer linear programming context, sequentially building up the solution along each branch enables one to keep vertex selection undetermined which in turn should limit the size of the search tree. Each node in the tree corresponds to a partial group sequence $\sigma = (V_1, \ldots, V_r)$. The subproblem at this node - the PCGTSP where any feasible tour is constrained to begin with the group sequence σ - is denoted by $\mathcal{P}(\sigma)$.

By applying dynamic programming, the minimum-cost paths between each pair of vertices in V_1 and V_r through σ may be computed incrementally. Based on this simple observation, we consider two different ways of defining a reduced PCGTSP which could be used for computing lower bounds for $\mathcal{P}(\sigma)$.

Definition 1. *Given a PCGTSP instance $\mathcal{P}$ and fixed sequence $\sigma = (V_1, \ldots, V_r)$ of already chosen groups, we define $\mathcal{P}_1(\sigma)$ as the PCGTSP instance with*

- *the same precedence constraints as $\mathcal{P}$,*
- *the same vertices and groups as $\mathcal{P}$ except for the inner groups of σ which are excluded,*
- *the same arc costs as $\mathcal{P}$ except for outgoing arcs from V_1 and incoming arcs to V_r,*
- *arc costs from vertices in V_1 to vertices in V_r given by the corresponding minimum path costs through σ, and*
- *remaining outgoing arcs from V_1 and incoming arcs to V_r are excluded.*

Proposition 1. $C_{\min}(\mathcal{P}(\sigma)) = C_{\min}(\mathcal{P}_1(\sigma))$

Proof. Follows from Definition 1. □

As a consequence, any PCGTSP bound may be applied to $\mathcal{P}_1(\sigma)$ to generate a bound for $\mathcal{P}(\sigma)$. The second definition is aimed at separating the cost of the path along σ from the cost of the rest of the tour.

Definition 2. *Given a PCGTSP instance $\mathcal{P}$ and a fixed sequence $\sigma = (V_1, \ldots, V_r)$ of already chosen groups, we define $\mathcal{P}_2(\sigma)$ as the PCGTSP instance with*

- *the same precedence constraints as $\mathcal{P}$,*
- *the same vertices and groups as $\mathcal{P}$ except for the inner groups of σ which are excluded,*
- *the same arcs costs as $\mathcal{P}$ except for outgoing arcs from V_1 and incoming arcs to V_r,*
- *arc costs from V_1 to V_r are set to zero, and*

- *remaining outgoing arcs from V_1 and incoming arcs to V_r are excluded.*

Let $c_{\min}(\sigma)$ be the cost of the shortest path through σ. Then the following holds:

Proposition 2.

$$C_{\min}(\mathcal{P}(\sigma)) \geq C_{\min}(\mathcal{P}_2(\sigma)) + c_{\min}(\sigma) \tag{1}$$

Proof. This follows since any tour of $\mathcal{P}(\sigma)$ may be split into one path along σ, and one path between V_r and V_1 obeying the precedence constraints. The former will have a cost that is at least $c_{\min}(\sigma)$ by definition. The latter will have the same cost as the $\mathcal{P}_2(\sigma)$ tour created by adding the proper arc between V_1 and V_r since all those arcs have zero costs. It then follows that this path will have a cost at least $C_{\min}(\mathcal{P}_2(\sigma))$ by Definition 2. □

Proposition 2 means that computing a lower bound for $\mathcal{P}_2(\sigma)$ and adding $c_{\min}(\sigma)$ will result in a valid lower bound for $\mathcal{P}(\sigma)$. The main reason for using this approach instead of that based on Defintion 1 is that many subproblems $\mathcal{P}_2(\sigma)$ will be identical for different nodes in the branching tree. This may be exploited to reduce computations and will be further explored in Section 3.5.

Equality in Proposition 2 is not guaranteed since the optimal $\mathcal{P}(\sigma)$ tour may have node selections for V_1 and V_r that do not match the optimal path for σ or the optimal $\mathcal{P}_2(\sigma)$ tour or both. Obviously, any PCGTSP bound may be applied to $\mathcal{P}_2(\sigma)$ to achive a lower bound on $\mathcal{P}(\sigma)$ by adding the cost of the cheapest path in σ.

We choose depth-first search as a branching strategy as this will rapidly deliver new upper bound estimates and is also less memory intensive compared to other strategies such as breadth-first or best-first. When selecting the next branch to explore among several possible at the same depth, we pick the group V_p with the largest corresponding successor set $\{q : (p, q) \in \Pi\}$. By prioritizing large successor sets, precedence constraints may be eliminated early in the branching, which should be beneficial considering that the bounding methods which are evaluated in this paper relax these constraints. Furthermore, a group with a particularly large successor set is likely to turn up early in an optimal tour. The hope is that one will obtain better upper bounds when using this priority rather than branching in some other order.

When a group sequence with m groups is reached, the vertex choice is optimized and a special 3-opt heuristic [16] is applied. If the solution value obtained is lower than that of the current best upper bound then the upper bound is updated.

3.2 Relaxation and Bounding

In the literature, lower bounds for ATSP have commonly been obtained by either relaxing the integrality requirements on the variables in an integer linear program and solving the resulting linear program (LP), or by relaxing some problem-specific constraints and then solving the resulting polynomial-time solvable problem.

The two most common relaxations of the second type result in either a minimum spanning arborescence problem (MSAP), or a minimum vertex-disjoint cycle cover problem, which is equivalent to the assignment problem (AP). The MSAP is obtained by relaxing the so-called outdegree constraints which dictate that each vertex should have only one outgoing arc, and the AP is obtained by relaxing the subtour elimination constraints (SEC) which ensure that only one cycle occurs in the ATSP solution.

When applying relaxations of this sort to the asymmetric GTSP, the resulting combinatorial problems are NP-hard. This was shown by Myung et al. for the generalized MSAP [27] and by Gutin and Yeo for the generalized AP [19]. Since this excludes polynomial-time algorithms for these problems, using them for bounding in a branch-and-bound scheme is impractical.

To circumvent this issue we can either transform the AGTSP to an equivalent ATSP (see [12,25,29]), or relax the vertex choice constraints which allow only one vertex to be visited in each group. The latter relaxation allows a solution to exit each group from any vertex regardless of where it entered. Therefore, only the minimum cost arcs between the groups are relevant in this relaxed problem (see [28]).

Incorporating the precedence constraints in the bounding procedure is a more sophisticated matter. These constraints have been modeled both as an exponential [13] and a polynomial [31] family of constraints, although the latter formulation requires auxiliary variables which complicate the model. In [13], Escudero et al. introduce a bounding scheme for the SOP where the precedence constraints are taken into account by introducing valid cuts based on the exponential family of constraints. However, because of the complexity in handling a potentially exponential family of cuts in the bounding procedure we will instead opt to drop the precedence constraints almost completely. In the following relaxed problems we

assume that if $(p, q) \in \Pi$ then all arcs (j, i) such that $i \in V_p$ and $j \in V_q$ are excluded from the graph, and otherwise ignore the precedence constraints. This GTSP without precedence constraints, which is defined on a possibly less connected graph, will be referred to as the *weak version* of the PCGTSP.

3.3 Minimum Spanning Arborescence Problem

Relaxing the outdegree constraints in the weak version of the PCGTSP will give rise to the generalized MSAP which requires a minimum cost directed tree (arborescence) such that exactly one vertex in each group is included. To obtain an ordinary MSAP we define a relaxed problem and a transformed problem which are defined on a non-partitioned vertex set.

Definition 3. *For any PCGTSP instance $\mathcal{P}$, let* $\mathrm{NC}(\mathcal{P})$ *be the ATSP instance defined on the graph* $G_{\mathrm{NC}} = (M, A_{\mathrm{NC}})$ *where* $A_{\mathrm{NC}} = \{(p, q) : p \in M, q \in M, (q, p) \notin \Pi\}$. *For every* $(p, q) \in A_{\mathrm{NC}}$ *the arc costs* c^{NC}_{pq} *are defined as* $c^{\mathrm{NC}}_{pq} = \min(\{c_{ij} : i \in V_p, j \in V_q\})$.

As shown in [28], $\mathrm{NC}(\mathcal{P})$ is equivalent to the weak GTSP version of $\mathcal{P}$ with the vertex choice constraint relaxed. Solving $\mathrm{NC}(\mathcal{P})$ provides a lower bound on $\mathcal{P}$ but it is NP-hard as it constitutes an ATSP with m vertices. However, a lower bound on $\mathrm{NC}(\mathcal{P})$ is obviously a lower bound on $\mathcal{P}$.

For the transformed problem we will make use of the Noon-Bean transformation. It defines zero cost arcs within every group such that its vertices are connected to form a directed cycle with some fixed order. Furthermore, if a cycle in a group V_p is given the order $v_1, v_2, \ldots, v_k$ then every outgoing arc cost $c_{v_i j}$, $j \in V_q : q \neq p$, is replaced by $c_{v_{i+1} j}$ (with v_k getting the outgoing arc costs of v_1).

Definition 4. *For any PCGTSP instance $\mathcal{P}$, let* $\mathrm{NB}(\mathcal{P})$ *denote the ATSP instance which arises when applying the Noon-Bean transformation [29] to the weak version of $\mathcal{P}$. The modified arc costs are denoted* c^{NB}_{ij}.

The problem $\mathrm{NB}(\mathcal{P})$ is to find a minimum-cost closed tour such that each vertex in V is visited exactly once using the modified arc costs c^{NB}_{ij}. Since $\mathrm{NB}(\mathcal{P})$ is an ATSP instance with n vertices with an optimal solution which is equal to that of the weak version of the original PCGTSP, any lower bound for this problem is also valid for the original PCGTSP.

Relaxing the outdegree constraints in $\mathrm{NB}(\mathcal{P})$ or $\mathrm{NC}(\mathcal{P})$ will result in an MSAP instance with n or m vertices respectively. To further tighten the bound we will also add the minimum-cost incoming arc to the root vertex of the arborescence. This is commonly done when bounding the ATSP with an MSA and is known as a *1-arborescence* [36].

3.4 Assignment Problem

By relaxing the subtour elimination constraints in $\mathrm{NC}(\mathcal{P})$ one obtains the cycle cover problem which asks to find pairwise vertex-disjoint directed cycles that together cover every vertex exactly once, and minimize the total cost of the arcs contained in these cycles. This problem can be equivalently formulated as the assignment problem (AP). Create two copies M' and M'' of M. The copies of every vertex $p \in M$ are denoted p' and p'' accordingly. Define a bipartite graph with bipartite sets M' and M''. For every arc (p, q) create the arcs (p', q'') and (p'', q'), having the same cost as (p, q). This yields a one-to-one correspondence between the cycle covers in the given graph and the assignments in the constructed bipartite graph.

Applying the same type of relaxation to $\mathrm{NB}(\mathcal{P})$ will potentially yield worse lower bounds as there will exist one zero cost cycle in its graph for every group with $|V_p| > 1$, $p \in M$. So it can be conjectured that the more groups containing more than a single vertex in an instance of $\mathcal{P}$, the weaker the bound will be.

We derive an alternative to $\mathrm{NC}(\mathcal{P})$ when computing the AP bound by introducing so called L-paths:

Definition 5. *In an instance of GTSP and for a fixed integer $L \geq 1$, an L-path is a path of exactly L arcs visiting $L + 1$ groups. Let $d_L(p, q)$ denote the length of a shortest L-path whose start vertex is in p and whose end vertex is in q. The L-th power of the GTSP instance is the directed graph with M as the vertex set and arcs (p, q) with costs $d_L(p, q)$. Let this be denoted* $\mathrm{NC}_L(\mathcal{P})$.

Note that $d_1(p, q)$ is simply the minimum cost of all arcs from p to q and the resulting problem corresponds to $\mathrm{NC}(\mathcal{P})$ defined in Definition 3 (i.e. $\mathrm{NC}(\mathcal{P}) = \mathrm{NC}_1(\mathcal{P})$). For every fixed $L > 1$ one can compute the $d_L(p, q)$ in polynomial time, by doing L steps of the Held-Karp dynamic programming method, but for arbitrary start vertices. Once the L-th power is generated, we can also compute a minimum-cost cycle cover therein in polynomial time (as in any directed graph).

Proposition 3. *The minimum cost of a cycle cover in the L-th power of a GTSP instance, divided by L, is a lower bound on the cost of any tour.*

Proof. Consider any tour through the groups, that is, any solution to GTSP. By re-indexing we assume that the tour is $(1, 2, \ldots, m)$. This tour contains the L-paths $(i, i+1, \ldots, i+L)$, where $i = 1, 2, \ldots, m$, and additions are meant modulo m (and m is used instead of 0). Clearly, every arc of the tour belongs to exactly L of these L-paths. Hence the cost of the tour equals $1/L$ times the sum of the costs of all these L-paths. Furthermore, every group in M is the start group and end group, respectively, of exactly one of these L-paths. In other words, the arcs $(i, i+L)$ form a cycle cover in the L-th power.

Now let us replace the actual cost of each subpath $(i, i+1, \ldots, i+L)$ of the considered tour with $d_L(i, i+L)$. By definition of the $d_L(p, q)$ this can only decrease the costs. We conclude that $1/L$ times the cost of the obtained cycle cover in the Lth power is a lower bound on the cost of the tour. Consequently, a minimum-cost cycle cover in the Lth power is a lower bound on the optimal GTSP solution as well. □

For ease of presentation the above reasoning was done for GTSP, that is, in the absence of precedence constraints. Trivially, the lower bound from Proposition 3 remains valid for PCGTSP, since further constraints can only raise the optimal tour costs. However we may obtain stronger lower bounds by taking precedence constraints into account already in the definition (and calculation) of the distances $d_L(p, q)$. In order to guarantee that every $d_L(p, q)$ remains smaller than or equal to the length of the corresponding L-path in the tour, the precedence constraints must be treated carefully. We re-define $d_L(p, q)$ as follows.

Definition 6. *Let $d_L(p, q)$ be the length of a shortest L-path P which has its start vertex in p and its end vertex in q, and satisfies the following conditions.*
Case $1 \notin P$: For every $i, j \in P$ such that $(g(i), g(j)) \in \Pi$, vertex i appears earlier than j in P.
Case $1 \in P$: Let P_0 be the subpath of P from p to 1, but excluding 1. Let P_1 be the subpath of P from 1 to q, including 1. For every $i, j \in P_0$ such that $(g(i), g(j)) \in \Pi$, vertex i appears earlier than j in P_0; and similarly for P_1. Furthermore, there is no $i \in P_0$ and $j \in P_1$ with $(g(i), g(j)) \in \Pi$.

These "precedence-aware" $d_L(p, q)$ can still be computed by dynamic programming, and the counterpart of Proposition 3 holds for them, because the proof literally goes through.

The rationale behind using L-paths with $L > 1$ can be described as follows. The bound for $L = 1$ is simple, but it totally ignores the requirement that the tour must enter and leave every group through the same vertex. As opposed to this, the inner vertices of an L-path satisfy this requirement. On the other hand, an inner vertex of an L-path from p to q may have a short distance to p or q, but it might not occur in the tour. Such constellations yield too small $d_L(p, q)$ values for the lower bound. We refer such inner vertices as "bad via-vertices". As a consequence, a larger L can make the lower bound stronger or weaker, depending on the instance. Intuitively, larger L should in general work better for larger group sizes.

3.4.1 Time Complexity

Fast computation of the $d_L(p, q)$ turns out to be crucial for the overall running time if the L-path bound is used. In the following we consider $L = 2$.

Define $d(p, v)$ as the length of a shortest arc from a group p to a vertex v, and define $d(v, q)$ similarly. Observe that $d_2(p, q) = \min_v d(p, v) + d(v, q)$, where v runs through all vertices outside the groups p and q. (Precedence constraints are easily taken into account: A directed distance is infinite if there is a precedence constraint in the opposite direction.)

Proposition 4. *With n and m being the total number of vertices and groups, respectively, we can compute all $d_2(p, q)$ in $O(n^2 + m^2 n)$ time.*

Proof. First compute all $d(p, v)$ and $d(v, q)$. This takes $O(n^2)$ time, as one must consider every vertex v and all adjacent vertices, and take the minima in all groups. Then apply the above formula for $d_2(p, q)$ using these precomputed values. It takes $O(m^2 n)$ time to consider all $O(m^2)$ pairs of groups and the n inner vertices v for each pair. The total time is $O(n^2 + m^2 n)$. □

Note that this time bound is subcubic in n, and only quadratic if we have a small number of large groups.

3.5 History Utilization

In this section we will describe a generalization of the history utilization pruning technique developed by Shobaki & Jamal in [32]. Since the vertex choice is never fixed during branching we are not able to extend every part of the technique but the principal idea is still applicable.

The technique revolves around mainly two ideas: (1) there is a high probability of solving identical bounding subproblems in the search tree, and (2) any optimal tour must include the shortest of all feasible paths which start at the start group and end at any vertex which is included in the optimal tour. Statement (2) is identical to the fact which enables dynamic programming for PCGTSP.

To describe the history utilization we first define equivalency between tree nodes.

Definition 7. *For every pair (S, r), $S \subseteq M$, $|S| > 1$, and $r \in S$, we define $T(S, r)$ to be the set of all tree nodes whose group sequences begin at V_1, traverse the groups in S (and no other groups), and end at group V_r. Any two tree nodes belonging to the same set $T(S, r)$ are said to be equivalent.*

In the remainder of this section we consider an unprocessed tree node $\mathcal{N}(\sigma) \in T(S, r)$ with partial group sequence $\sigma = (V_1, \ldots, V_r)$. Let $P_{ij}^{(\sigma)}$ be a shortest path between the vertex pair $i \in V_1$ to $j \in V_r$, through σ and let $c(P_{ij}^{(\sigma)})$ denote its costs. Also assume that $P_{ij}^{(S,r)}$ is the shortest path from node $i \in V_1$ to node $j \in V_r$ which has been discovered during the branch-and-bound search, with $c(P_{ij}^{(S,r)})$ denoting its cost. If no tree node in $T(S, r)$ has been processed then $c(P_{ij}^{(S,r)}) = \infty$.

Proposition 5. *If $c(P_{ij}^{(S,r)}) < c(P_{ij}^{(\sigma)}), \forall (i, j) \in V_1 \times V_r$ then there cannot exist a solution to the PCGTSP which begins with the group sequence σ and has smaller total cost than a solution which begins with $P_{ij}^{(S,r)}$, $\forall (i, j) \in V_1 \times V_r$. In other words, the tree node $\mathcal{N}(\sigma)$ can be pruned.*

Proof. Take $P_{ij}^{(S,r)}$ and $P_{ij}^{(\sigma)}$, $(i, j) \in V_1 \times V_r$, with $c(P_{ij}^{(S,r)}) < c(P_{ij}^{(\sigma)})$. Let the best solution which includes $P_{ij}^{(S,r)}$ have total cost $z_{\text{opt}}^{(S,r)}$ and the best solution which includes $P_{ij}^{(\sigma)}$ have total cost $z_{\text{opt}}^{(\sigma)}$. Now assume that $z_{\text{opt}}^{(\sigma)} < z_{\text{opt}}^{(S,r)}$. Then, since $c(P_{ij}^{(S,r)}) < c(P_{ij}^{(\sigma)})$, we must have that

$$z_{\text{opt}}^{(\sigma)} - c(P_{ij}^{(\sigma)}) < z_{\text{opt}}^{(S,r)} - c(P_{ij}^{(S,r)}) \iff z_{\text{opt}}^{(\sigma)} - c(P_{ij}^{(\sigma)}) + c(P_{ij}^{(S,r)}) < z_{\text{opt}}^{(S,r)} \tag{2}$$

The solution which includes σ includes a partial path (the complement of $P_{ij}^{(\sigma)}$ in the solution) $\hat{\sigma} = (j, v_{L+1}, v_{L+2}, \ldots, v_m, i)$ with cost $z_{\text{opt}}^{(\sigma)} - c(P_{ij}^{(\sigma)})$. Since $P_{ij}^{(S,r)}$ and $P_{ij}^{(\sigma)}$ start and end at the same vertices we may concatenate $P_{ij}^{(S,r)}$ and $\hat{\sigma}$ to form a valid solution with cost $z_{\text{opt}}^{(\sigma)} - c(P_{ij}^{(\sigma)}) + c(P_{ij}^{(S,r)})$. But $z_{\text{opt}}^{(\sigma)} - c(P_{ij}^{(\sigma)}) + c(P_{ij}^{(S,r)}) < z_{\text{opt}}^{(S,r)}$ according to (2), which contradicts the fact that $z_{\text{opt}}^{(S,r)}$ is the cost corresponding to the shortest possible solution which includes $P_{ij}^{(S,r)}$. □

If it is not possible to prune a tree node based on already explored similar nodes we can forgo solving the bounding subproblem and reuse information from an already explored tree node. This can be done when using the subproblem definition $\mathcal{P}_2(\sigma)$ since the lower bound $z_{\text{LB}}^{(\sigma)}$ in each tree node is separated into two parts: the prefix, $z_{\text{prefix}}^{(\sigma)}$, which is the cost of the minimum-cost path through σ, and the suffix, $z_{\text{suffix}}^{(\sigma)}$, which is a lower bound on $\mathcal{P}_2(\sigma)$. Note that we have that the suffix part of the lower bound is equal for all similar tree nodes:

$$z_{\text{suffix}}^{(\sigma)} = z_{\text{suffix}}^{(S,r)}, \qquad \forall \mathcal{N}(\sigma) \in T(S, r).$$

If $c(P_{ij}^{(\sigma)}) < c(P_{ij}^{(S,r)})$ for some $(i, j) \in V_1 \times V_r$ then $\mathcal{N}(\sigma)$ cannot be pruned according to Proposition 5. However, if another node in $T(S, r)$ has been processed and $z_{\text{suffix}}^{(S,r)}$ has been stored, then $z_{\text{LB}}^{(\sigma)}$ can be obtained in $O(1)$ time by computing

$$z_{\text{LB}}^{(\sigma)} = \min_{(i,j) \in V_1 \times V_L} (c(P_{ij}^{(\sigma)})) + z_{\text{suffix}}^{(S,r)}.$$

3.6 Held-Karp Dynamic Programming

The dynamic programming approach to solving the PCGTSP is described in [8]. It is very memory intensive and exponential in its time complexity. However, small instances ($m < 20$) can be solved fairly quickly on a normal computer using dynamic programming and because of this it is included and evaluated as a sort of "presolve" step in our algorithm.

3.7 Summary of Algorithm

A pseudo-code outline of the algorithm, and the branching and bounding subroutines is given below.

Algorithm 3.1 outlines the general procedure for the solver. At line 6 the algorithm checks if the instance is small enough to solve it directly with dynamic programming. The while-loop at line 13 makes sure that the algorithm evaluates all nodes in the search tree that are not pruned. The if-statements inside the loop checks if the node being evaluated is a complete solution and if it should be saved, or if the node should be branched or pruned.

Algorithm 3.1 Branch-and-bound for the PCGTSP

```
 1: function SOLVE(𝒫)
 2:     LB_best = 0
 3:     UB_best = ∞
 4:     bestLeaf = NULL
 5:     Let Q be an empty stack
 6:     if m < 20 then
 7:         DynamicProgramming(𝒫)
 8:     else
 9:         Let 𝒩_0 be the root node
10:         Bound(𝒩_0)                                   ▷ Bound the root node
11:         LB_best = 𝒩_0.LB
12:         Q.push(𝒩_0)
13:         while Q is not empty do
14:             currentNode = Q.pop()
15:             if currentNode.IsLeaf & currentNode.UB < UB_best then
16:                 bestLeaf = currentNode
17:                 UB_best = currentNode.UB
18:             else if currentNode.LB < UB_best then
19:                 Branch(Q, currentNode)                ▷ Branch node
20:             else
21:                 delete currentNode                    ▷ Prune node
22:             end if
23:         end while
24:     end if
25: end function
```

The bounding subroutine outlined in Algorithm 3.2 simply takes a tree node and computes the bound for the corresponding subproblem. Line 6 checks if some other tree node belonging to $T(S(\sigma), p)$ has been evaluated before the current node and picks out the lower bound for the subproblem directly from a hash map. Otherwise the lower bound is computed using some fixed method. The bounding method used depends on the Boolean parameters "useArborescence" and "useNoonBean".

Algorithm 3.3 creates new tree nodes and adds them to the queue of nodes to be evaluated in Algorithm 3.1. Line 7 enforces the branching priority where groups with a high number of successors are evaluated first. Since the queue of nodes to be evaluated acts as a stack, the groups with the smallest number of successors are pushed first. Line 11 checks if the resulting tree node is a complete solution. Line 14 checks if the node can be pruned directly by utilizing Proposition 5. Line 17 creates the appropriate subproblem $\mathcal{P}_2(\sigma)$ according to Definition 2. Line 19 updates the shortest path costs $P_{ij}^{(S,r)}$.

Algorithm 3.2 Bounding subroutine

function BOUND($\mathcal{N}$)
  Let historySuffix[] be a global hashmap ▷ Suffix part of lower bound for every (S, r) pair
  $\sigma = \mathcal{N}$.GroupSequence
  Let $S(\sigma)$ be the set of groups in σ
  Let p be the last group in σ
  if historySuffix$[(S(\sigma), p)] < \infty$ **then**
    lowerBound = historySuffix$[(S(\sigma), p)]$
  else
    if useNoonBean **then**
      if useArborescence **then**
        lowerBound = **ComputeMinArborescence**(NB($\mathcal{N}$.SubProblem))
      else
        lowerBound = **ComputeMinAssignment**(NB($\mathcal{N}$.SubProblem))
      end if
    else
      if useArborescence **then**
        lowerBound = **ComputeMinArborescence**(NC_1($\mathcal{N}$.SubProblem))
      else
        lowerBound = **ComputeMinAssignment**(NC_L($\mathcal{N}$.SubProblem))
      end if
    end if
    historySuffix$[(S(\sigma), p)]$ = lowerBound
  end if
  $\mathcal{N}$.LB = $\min(P_{ij}^{(\sigma)})$ + lowerBound
end function

Algorithm 3.3 Branching subroutine

function BRANCH(Q, $\mathcal{N}$)
  Let historyPathCosts[] be a global hashmap ▷ Path costs for every (S, r) pair
  Let N be an empty stack
  currentGroupSequence = $\mathcal{N}$.GroupSequence
  allowedGroups = $\mathcal{N}$.AllowedGroups ▷ Contains all groups which are allowed to be sequenced after the last group in currentGroupSequence
  while allowedGroups is not empty **do**
    p = **FindSmallestSuccessorSet**(allowedGroups)
    σ = [currentGroupSequence, p]
    Let $\mathcal{N}(\sigma)$ be a new tree node
    Let $S(\sigma)$ be the set of groups in σ
    if $|S(\sigma)| = m$ **then**
      $\mathcal{N}(\sigma)$.IsLeaf = **true**
      $\mathcal{N}(\sigma)$.UB = **ComputeUpperBound**(σ) ▷ Applies three-opt heuristic to σ and computes the cost of the resulting tour
    else if historyPathCosts$[(S(\sigma), p)]_{ij} < c(P_{ij}^{(\sigma)})$,$\forall (i, j) \in V_1 \times V_p$ **then**
      delete $\mathcal{N}(\sigma)$ ▷ Prune node
    else
      $\mathcal{N}(\sigma)$.SubProblem = $\mathcal{P}_2(\sigma)$
      Bound($\mathcal{N}(\sigma)$)
      for every $(i, j) \in V_1 \times V_p$ **do**
        if historyPathCosts$[(S(\sigma), p)]_{ij} > c(P_{ij}^{(\sigma)})$ **then**
          historyPathCosts$[(S(\sigma), p)]_{ij} = c(P_{ij}^{(\sigma)})$
        end if
      end for
      Q.push($\mathcal{N}(\sigma)$)
    end if
    delete p in allowedGroups
  end while
end function

4 Results

We evaluate the algorithm and the different bounding methods on both synthetic and industrial problem instances. The synthetic instances are generated by taking SOP instances and randomly adding between 0 and 8 duplicates of each vertex except the starting vertex, and duplicating all of the corresponding arcs and precedence constraints. Each set of duplicate vertices and their original vertex are assigned a group number (where every vertex in the group have identical arc costs). Every arc cost is then multiplied by a random number drawn from the uniform distribution on the interval $[0.8, 1.0]$, giving each vertex its own set of arc costs. The synthetic instances are named "020.X", where "X" is the name of the SOP instance from which it has been derived, while the industrial problem instances - which are derived from the problem of coordinate measuring machine sequencing [30] - are named "cmm00x". All tests are run on a PC equipped with an Intel i7-6700K 4.00GHz CPU and 32GB of RAM, and are terminated after 24 hours. We also limit the maximum memory usage by the algorithm to 8GB since the hash maps which store subproblem bounds and shortest paths can grow uncontrollably if the algorithm is not able to sufficiently limit the search tree.

We first test the different bounding methods on a limited set of problem instances in order to choose the best one. The dynamic programming presolve step is disabled in order to evaluate the bounding methods on the smaller problem instances as well. The results can be seen in Table 4. We evaluate the different methods by comparing the upper and lower bound obtained when the algorithm terminates - either due to solving the problem or exceeding the time limit. We also assess the bounding methods' abilities to limit the search tree by comparing the number of tree nodes processed. The optimality gap is calculated by taking $(\text{UB} - \text{LB})/\text{UB}$. Given a PCGTSP instance $\mathcal{P}$, the different bounding methods are named as follows:

- AP-Lx - Assignment Problem solved over the instance $\text{NC}_\text{x}(\mathcal{P})$.
- AP-NB - Assignment Problem solved over the instance $\text{NB}(\mathcal{P})$.
- MSAP-L1 - Minimum Spanning Arborescence Problem solved over the instance $\text{NC}_1(\mathcal{P})$.
- MSAP-NB - Minimum Spanning Arborescence Problem solved over the instance $\text{NB}(\mathcal{P})$.

Table 1: Comparison of bounding methods. Dynamic programming presolve step is disabled.

Instance	Measure	AP-L1	AP-L2	AP-L3	AP-NB	MSAP-L1	MSAP-NB
cmm001	UB	49.12	49.12	49.12	49.12	49.12	49.12
($m = 12$)	LB	49.12	49.12	49.12	49.12	49.12	49.12
($n = 14$)	Gap	0.000	0.000	0.000	0.000	0.000	0.000
	# of nodes	194	129	91	2412	869	1078
	Time (s)	0.01	0.02	0.02	0.03	0.02	0.03
cmm002	UB	20.26	20.26	20.26	20.26	20.26	20.26
($m = 15$)	LB	20.26	20.26	20.26	20.26	20.26	20.26
($n = 24$)	Gap	0.000	0.000	0.000	0.000	0.000	0.000
	# of nodes	24633	25547	24274	63070	38555	38878
	Time (s)	0.16	0.18	0.28	0.29	0.23	0.27
cmm003	UB	20.04	20.04	20.04	20.04	20.04	20.04
($m = 17$)	LB	20.04	20.04	20.04	20.04	20.04	20.04
($n = 35$)	Gap	0.000	0.000	0.000	0.000	0.000	0.000
	# of nodes	5693	6285	6080	8400	7415	7707
	Time (s)	0.07	0.08	0.20	0.12	0.10	0.12
020.ESC12	UB	1389.77	1389.77	1389.77	1389.77	1389.77	1389.77
($m = 13$)	LB	1389.77	1389.77	1389.77	1389.77	1389.77	1389.77
($n = 64$)	Gap	0.000	0.000	0.000	0.000	0.000	0.000
	# of nodes	6351	10332	36518	116087	8093	8093
	Time (s)	0.24	0.47	4.79	1.60	0.29	0.91
020.ESC25	UB	1383.12	1383.12	1383.12	1383.12	1383.12	1383.12
($m = 26$)	LB	1383.12	1383.12	1383.12	0.00	1383.12	1383.12
($n = 134$)	Gap	0.000	0.000	0.000	1.000	0.000	0.000
	# of nodes	47150	570135	5326456	2237494611	1504659	1504659
	Time (s)	13.28	233.97	30825.32	86400.00	127.54	861.81
020.p43.4	UB	66848.40	66848.40	66848.40	66848.40	66848.40	66848.40
($m = 43$)	LB	66848.40	66848.40	66848.40	66848.40	66848.40	66848.40
($n = 205$)	Gap	0.000	0.000	0.000	0.000	0.000	0.000
	# of nodes	150442479	153568124	173404410	173240786	172191626	172191622
	Time (s)	3416.38	3633.95	31847.6	13580.93	3503.74	4141.19
020.ft53.4	UB	11823.20	11823.20	12530.62	12048.03	11823.20	11823.20
($m = 53$)	LB	11823.20	11823.20	6048.93	7.36	11823.20	11823.20
($n = 276$)	Gap	0.000	0.000	0.517	0.999	0.000	0.000
	# of nodes	242706376	244617487	20956149	282228333	348785599	348785617
	Time (s)	10152.13	12785.81	86400.00	86400.00	11360.23	18201.22
Averages	Gap	0.000	0.000	0.074	0.286	0.000	0.000
	# of nodes	**56176125**	56971148	**28536283***	384736242	74643159	74648236
	Time (s)	**1940.32**	2379.21	21296.89	26626.14	2141.74	3315.08

* Method has the lowest average of processed tree nodes but could not solve 020.ft53.4

All bounding methods are able to solve all problem instances within the 24 hour time limit except for AP-L3 and AP-NB. The AP-NB method is by far the slowest and weakest in terms of being able to prune tree nodes. The two MSAP methods do not differ much in terms of strength but the MSAP-L1 method is much faster since it solves the MSAP on a significantly smaller graph. AP-L1 is on average the strongest and fastest method for every problem instance in Table 4. Increasing the value of L does not seem to consistently strengthen the bound on the data that has been tested, and when $L = 3$ the bound computations are too slow to solve 020.ft53.4 within 24 hours. However, it should be noted that the SOP instances ESCxx are particularly ill-conditioned for $L > 1$ as the starting vertex is a sort of dummy-vertex which has zero incoming and outgoing arc costs to and from all other vertices and is therefore used as a cheap via-vertex for many L-paths.

We proceed by choosing AP-L1 as the bounding method and test the algorithm on instances which are larger and/or less dense with precedence constraints. The results are presented in Table 4. To get a

better measure of how close the upper bound is to optimality we also compute the gap (denoted Gap^{B}) when it is compared with the best known lower bound for the problem instance.

Table 2: Results when using AP bound with $\text{NC}_1(\mathcal{P})$. Dynamic programming presolve step is enabled.

Instance	m	n	$\lvert\Pi\rvert$	AP-L1					Best known	
				UB	LB	Gap	Gap^{B}	Time (s)	UB	LB
cmm001	12	14	36	49.12	49.12	0.000	0.000	0.01	49.12	49.12
cmm002	15	24	59	20.26	20.26	0.000	0.000	0.03	20.26	20.26
cmm003	17	35	81	20.04	20.04	0.000	0.000	0.06	20.04	20.04
cmm004	90	215	753	49.71	17.29	0.652	0.537	86400.00	46.07	23.00
cmm005	173	404	1210	219.99	73.95	0.664	0.664	86400.00	185.83	73.95
020.br17.10	18	89	31	44.28	44.28	0.000	0.000	9.82	44.28	44.28
020.br17.12	18	93	38	44.09	44.09	0.000	0.000	4.58	44.09	44.09
020.ESC12	13	64	23	1389.77	1389.77	0.000	0.000	0.16	1389.77	1389.77
020.ESC25	26	134	36	1383.12	1383.12	0.000	0.000	13.28	1383.12	1383.12
020.ESC47	48	245	79	1062.62	746.37	0.298	0.030	86400.00	1062.62	1030.40‡
020.ESC63	64	350	296	50.35	44.05	0.125	0.015	86400.00	50.35	49.60‡
020.ESC78	79	414	361	15463.80	7535.11	0.513	0.254	86400.00	14425.00†	11540.00‡
020.ft53.1	53	282	64	6197.41	4829.41	0.221	0.028	86400.00	6197.41	6024.80‡
020.ft53.2	53	275	82	6717.82	4854.96	0.277	0.044	86400.00	6717.82	6420.80‡
020.ft53.3	53	282	269	8718.28	4926.57	0.435	0.058	86400.00	8718.28	8209.60‡
020.ft53.4	53	276	811	11823.20	11823.20	0.000	0.000	10152.13	11823.20	11823.20
020.ft70.1	70	346	86	33955.72	30830.73	0.092	0.074	86400.00	33955.72	31450.40‡
020.ft70.2	70	351	117	35763.70	30992.80	0.133	0.103	86400.00	35763.70	32080.80‡
020.ft70.3	70	347	284	39081.12	31522.64	0.193	0.129	86400.00	39081.12	34028.00‡
020.ft70.4	70	353	1394	46722.84	34849.70	0.254	0.083	86400.00	46722.84	42824.00‡
020.p43.1	43	204	53	22649.90	602.29	0.973	0.006	86400.00	22649.90	22512.00‡
020.p43.2	43	199	76	22854.10	608.82	0.973	0.003	86400.00	22854.10	22784.00‡
020.p43.3	43	212	138	44926.40	607.45	0.986	0.487	86400.00	28835.00†	23068.00‡
020.p43.4	43	205	538	66848.40	66848.40	0.000	0.000	3416.38	66848.40	66848.40
020.ry48p.1	48	256	59	13151.45	10240.78	0.221	0.039	86400.00	13151.45	12644.00‡
020.ry48p.2	48	250	73	13804.57	10198.70	0.261	0.068	86400.00	13804.57	12859.20‡
020.ry48p.3	48	254	179	16949.32	10415.22	0.386	0.080	86400.00	16949.32	15592.00‡
020.ry48p.4	48	249	643	25980.00	25980.00	0.000	0.000	3555.31	25980.00	25980.00

† Upper bound obtained by taking best known SOP instance upper bound
‡ Lower bound obtained by multiplying SOP instance lower bound by max perturbation

Only three instances (020.br17.10, 020.br17.12, and 020.ry48p.4) other than the ones in Table 4 are solved within the time limit. For as many as 12/18 unsolved instances the algorithm produces upper bounds which are verified to be within at least 10% of the optimal solution. However, the lower bound produced at the root of the search tree is mostly too weak to prove this. Even though the algorithm is able to solve the densest version of the 020.ft53.x, 020.p43.x, and 020.ry48p.x instances, the amount of precedence constraints does not seem to be a deciding factor in terms of the quality of the produced solutions in the unsolved instances. This is a known result for the SOP where algorithms in general struggle the most with instances with precedence graphs which are half-way dense [26]. The dynamic programming presolve seems to be consistently faster for the instances with $m < 20$.

5 Discussion and Future Work

The results show the efficiency of different bounding methods and transformations for the PCGTSP. The experiments indicate that the Noon-Bean transformation is less effective than the node choice relaxation and the AP bound is marginally stronger than the MSAP bound.

While it seems like the assignment problem bound is not consistently strengthened when $L > 1$, it might be possible to utilize this idea better by modifying the branching rules. If one identifies a vertex which is especially cheap to visit, one can branch on this vertex such that one subproblem includes the vertex and the other excludes it. This can strengthen the bound in branches where cheap vertices are excluded while still exploring solutions where they are included. Furthermore, preliminary tests indicate that $L = 1$ is not consistently stronger than $L = 2$ for all subproblems in the search tree. Therefore, a hybrid method where one computes the bound for both $L = 1$ and $L = 2$ and then chooses the best

one may be a good approach. The current implementation of computing the distances for $L = 3$ has a computation time of $O(n^2 + m^2n^2)$ and is too slow to be practical. Developing this implementation coupled with a method to counteract the bad via-vertex phenomenon may enable a stronger AP bound.

Since the lower bound is never updated during the branch-and-bound algorithm unless the problem instance is solved to optimality, the bound at the root is especially important to strengthen in order to verify the quality of the produced solution. To strengthen the lower bound one could employ Lagrangian relaxation of the node choice constraints, and the vertex choice constraints or subtour elimination constraints depending on which method that is chosen. Some sort of nested subgradient method which updates the multipliers for the dualized constraints could then strengthen the lower bounds, and particular focus could be given to the root problem.

One could also explore the idea of branching on vertices and compare that to the branching method presented here. Even though branching on vertices may potentially increase the size of the search tree (due to the fact that $n \geq m$), fixed vertex choice may strengthen the subproblem bounds, and enable a more extended generalization of the history utilization pruning method.

One of the bigger drawbacks of the algorithm presented here is that the precedence constraints are almost completely relaxed and not taken into account when bounding the subproblems. One could consider formulating an ILP model and solving an LP relaxed problem which takes the precedence constraints into account somewhat. Tests in [30] indicate that general ILP solvers such as CPLEX take far too long to solve the LP relaxation and produces bounds which are not significantly stronger, indicating that some cutting plane augmentation is needed. However, this may be an issue with the mixed ILP model which was formulated and utilized for these tests. Another approach may be to develop valid precedence constraint cuts for the AP and/or MSAP in the same vein as in [13].

References

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

[2] N. Ascheuer, *Hamiltonian path problems in the on-line optimization of flexible manufacturing systems*, PhD Thesis, Tech. Univ. Berlin, 1995.

[3] N. Ascheuer, M. Jünger, G. Reinelt, *Heuristic algorithms for the asymmetric traveling salesman problem with precedence constraints — a computational comparison*, Technical Report, ZIB, Berlin (1998).

[4] N. Ascheuer, M. Jünger, G. Reinelt, *A Branch & Cut Algorithm for the Asymmetric Traveling Salesman Problem with Precedence Constraints*, Computational Optimization and Applications 17 (2000), pp. 61-84.

[5] E. Balas, *New classes of efficiently solvable generalized Traveling Salesman Problem*, Annals of Operations Research 86 (1999), pp 529-558.

[6] E. Balas, M. Fischetti, W.R. Pulleyblank, *The precedence-constrained asymmetric traveling salesman problem*, Mathematical Programming 68 (1995), pp. 241-265.

[7] K. Castelino, R. D'Souza, P.K. Wright, *Toolpath optimization for minimizing airtime during machining*, Journal of Manufacturing Systems 22(3) (2003), pp. 173-180.

[8] A. Chentsov, M, Khachay, D. Khachay, *Linear time algorithm for Precedence Constrained Asymmetric Generalized Traveling Salesman Problem*, IFAC-PapersOnLine 49(12) (2016), pp. 651-655.

[9] A.A. Ciré and WJ. van Hoeve, *Multivalued Decision Diagrams for Sequencing Problems*, Operations Research 61(6) (2013), pp. 1411-1428.

[10] R. Dewil, P. Vansteenwegen, D. Cattrysse, *Construction heuristics for generating tool paths for laser cutters*, International Journal of Production Research 52(20) (2014), pp. 5965-5984.

[11] R. Dewil, P. Vansteenwegen, D. Cattrysse, M. Laguna, T. Vossen, *An improvement heuristic framework for the laser cutting tool path problem*, International Journal of Production Research 53(6) (2015), pp. 1761-1776.

[12] V. Dimitrijević, *An efficient transformation of the generalized traveling salesman problem into the traveling salesman problem on digraphs*, Information Sciences 102(1-4) (1997), pp. 105-110.

[13] L.F. Escudero, M. Guignard, K. Malik, *A Lagrangian relax-and-cut approach for the sequential ordering problem with precedence relationships*, Annals of Operations Research 50 (1994), pp. 219-237.

[14] M. Fischetti, J.J. Salazar Gonsalez, P. Toth, *The symmetric generalized traveling salesman polytope*, NETWORKS 26(2) (1995), pp. 113–123.

[15] M. Fischetti, J.J. Salazar Gonsalez, P. Toth, *A Branch-And-Cut Algorithm for the Symmetric Generalized Traveling Salesman Problem*, Operations Research 45(3) (1997), pp. 378–394.

[16] L.M Gambardella and M. Dorigo, *An Ant Colony System Hybridized with a New Local Search for the Sequential Ordering Problem*, INFORMS Journal on Computing 12(3) (2000), pp 237-255.

[17] L. Gouveia and M. Ruthmair, *Load-dependent and precedence-based models for pickup and delivery problems*, Computers & Operations Research 63 (2015), pp. 56-71.

[18] G. Gutin, D. Karapetyan, N. Krasnogor, *A memetic algorithm for the generalized traveling salesman problem*, Natural Computing 9 (2010), pp. 47–60.

[19] G. Gutin and A. Yeo, *Assignment problem based algorithms are impractical for the generalized TSP*, Australasian Journal of Combinatorics 27(1) (2003), pp. 149-153.

[20] M. Held and R.M. Karp, *A Dynamic Programming Approach to Sequencing Problems*, Journal of the Society for Industrial and Applied Mathematics 10:1 (1962), pp 196-210.

[21] K. Helsgaun, *Solving the equality generalized traveling salesman problem using the Lin–Kernighan–Helsgaun Algorithm*, Mathematical Programming Computation 7(3) (2015), pp. 269-287.

[22] I.T. Hernádvölgyi, *Solving the Sequential Ordering Problem with Automatically Generated Lower Bounds*, Operations Research Proceedings 2003, pp 355-362.

[23] D. Karapetyan and G. Gutin, *Lin–Kernighan heuristic adaptations for the generalized traveling salesman problem*, European Journal of Operational Research 208(3) (2011), pp. 221–232.

[24] G. Laporte, H. Mercure, Y. Nobert, *Generalized Travelling Salesman Problem Through n Sets Of Nodes: The Asymmetrical Case*, Discrete Applied Mathematics 18 (1987), pp. 185-197.

[25] YN. Lien, *Transformation of the generalized traveling-salesman problem into the standard traveling-salesman problem*, Information Sciences 74(1-2) (1993), pp. 177-189.

[26] R. Montemanni, M. Mojana, G.A. Di Caro, L.M. Gambardella, *A decomposition-based exact approach for the sequential ordering problem*, Journal of Applied Operational Research 5(1) (2013), pp 2-13.

[27] YS. Myung, CH. Lee, DW. Tcha, *On the generalized minimum spanning tree problem*, Networks 26(4) (1995), pp. 231-241.

[28] C.E. Noon and J.C. Bean, *A Lagrangian Based Approach for the Asymmetric Generalized Traveling Salesman Problem*, Operations Research 39(4) (1991), pp. 623-632.

[29] C.E. Noon and J.C. Bean, *An Efficient Transformation Of The Generalized Traveling Salesman Problem*, INFOR: Information Systems and Operational Research 31(1) (1993), pp. 39-44.

[30] R. Salman, J.S. Carlson, F. Ekstedt, D. Spensieri, J. Torstensson, R. Söderberg, *An industrially validated CMM inspection process with sequence constraints*, Procedia CIRP Volume 44 (2016), pp. 138-143.

[31] S.C. Sarin, H.D. Sherali, A. Bhootra, *New tighter polynomial length formulations for the asymmetric traveling salesman problem with and without precedence constraints*, Operations Research Letters 3(1) (2005), pp. 62-70.

[32] G. Shobaki and J. Jamal, *An exact algorithm for the sequential ordering problem and its application to switching energy minimization in compilers*, Computational Optimization and Applications 61(2) (2015), pp 343-372.

[33] L.V. Snyder and M.S. Daskin, *A random-key genetic algorithm for the generalized traveling salesman problem*, European Journal of Operational Research 174 (2006), pp. 38–53.

[34] S.L. Smith and F. Imeson, *GLNS: An effective large neighborhood search heuristic for the Generalized Traveling Salesman Problem*, Computers & Operations Research 87 (2017), pp. 1-19.

[35] J. Sung and B. Jeong, *An Adaptive Evolutionary Algorithm for the Traveling Salesman Problem with Precedence Constraints*, The Scientific World Journal Volume 2014 (2014).

[36] D.P. Williamson, *Analysis of the Held-Karp lower bound for the asymmetric TSP*, Operations Research Letters 12(2) (1992), pp 83-88.