THESIS FOR THE DEGREE OF LICENTIATE OF PHILOSOPHY

Optimizing and Approximating Algorithms for the Single and Multiple Agent Precedence Constrained Generalized Traveling Salesman Problem

Raad Salman

Department of Mathematical Sciences
Division of Applied Mathematics and Statistics
Chalmers University of Technology and University of Gothenburg
Gothenburg, Sweden 2017

Optimizing and Approximating Algorithms for the Single and Multiple Agent Precedence Constrained Generalized Traveling Salesman Problem
Raad Salman

Department of Mathematical Sciences
Division of Applied Mathematics and Statistics
Chalmers University of Technology and University of Gothenburg
SE-412 96 Gothenburg, Sweden
Phone: +46 (0)31-772 10 00

Fraunhofer-Chalmers Research Centre for Industrial Mathematics
Department of Geometry and Motion Planning
Chalmers Science Park
SE-412 88 Gothenburg, Sweden
Phone: +46 (0)31-772 4254
Email: raad.salman@fcc.chalmers.se

Printed in Gothenburg, Sweden 2017

In loving memory of Hassan Salman.

Abstract

In the planning phases of automated manufacturing, generating efficient programs for robot stations is a crucial problem which needs to be solved. One aspect of the programming is the optimization of task sequences, such as series of welds or measuring points, so that the cycle time is minimized.

This thesis considers the task sequencing problem separately from the other problems related to robot station programming such as motion planning and collision avoidance. The stations may have a single robot or an arbitrary (but predetermined) number of robots. The robots are heterogeneous with respect to their ability to perform the different tasks, and may have several movements and angles to choose from when performing a task. Furthermore, some processes require that the tasks are performed within some partial order. This may cause delays when there are more than one robot at a station. The task sequencing problem is then modeled as the Precedence Constrained Generalized Multiple Traveling Salesman Problem.

A metaheuristic algorithm based on Ant Colony Optimization is considered in conjunction with several different local search heuristics. The local search neighborhoods are analyzed with respect to the notion of improvement and induced delays due to precedence constraints between robots. In the single robot case, the HACS algorithm is shown to find solutions at least within 10% of the optimum on average, often within a couple of seconds. For stations with multiple robots, the results indicate that the local search procedure is good at improving solutions but that it becomes very computationally demanding when applied to instances with a large number of precedence constraints.

Additionally, an exact branch-and-bound based algorithm is presented for single robot stations. A novel branching method is developed, a pruning technique used for a related problem is generalized, and a new way of computing an assignment problem based bound is evaluated. The algorithm is able to solve some medium sized problem instances (around 50 tasks) within 24 hours. Many of the smaller problem instances are solved within seconds.

Keywords: asymmetric generalized multiple traveling salesman problem; precedence constraints; sequential ordering problem; vehicle routing problem; metaheuristic, local search heuristic; ant colony optimization; edge exchange; branch and bound; dynamic programming; SOP; GTSP; mTSP; VRP; PCGTSP; PCGmTSP

Acknowledgments

First and foremost, I would like to thank my supervisors Fredrik Ekstedt and Peter Damaschke for all the rewarding discussions and guidance during this thesis work. Thanks to Fraunhofer-Chalmers Centre (FCC) and the Department of Mathematical Sciences at Chalmers for giving me the opportunity to work on this project. I would also like to extend a special thanks to Domenico Spensieri, Johan S. Carlson, and everyone else at FCC who has supported me with their involvement.

Finally, thanks to all of my friends and especially my family whose love and support has been indispensable during these last two years.

Raad Salman
Göteborg, October 2017

List of Publications

- **Paper I**. R. Salman, J.S. Carlson, F. Ekstedt, D. Spensieri, J. Torstensson, R. Söderberg, *An industrially validated CMM inspection process with sequence constraints*, Procedia CIRP (2016), pp. 138–143.
- **Paper II**. R. Salman, F. Ekstedt, P. Damaschke, *Branch-and-bound for the Precedence Constrained Generalized Traveling Salesman Problem*, Submitted (2017).
- **Paper III**. F. Ekstedt, R. Salman, D. Spensieri, *A Hybridized Ant Colony System Approach to the Precedence Constrained Generalized Multiple Traveling Salesman Problem*, Manuscript (2017).

Contents

1. Introduction

1.1 Background

As computer aided product development and manufacturing are increasingly becoming the norm, efficient programs which reduce the use of resources such as energy and material are now more important than ever. From design to planning, and finally manufacturing, numerous mathematical problems must be overcome when creating a robust computer aided process for product realization.

The software Industrial Path Solutions (IPS) is a virtual simulation tool capable of modeling and optimizing many aspects of automated manufacturing processes. Such a process is the generation of efficient robot programs where the problem of task sequencing naturally arises. The task sequencing problem consists of determining the order of tasks on a robot station such that the total time for finishing the tasks (known as the cycle time) is minimized. The stations may have several robots working on the same object simultaneously and be such that the robots have either shared or unshared workspaces.

In the IPS software, paths and movements for the robots are computed separately from the task sequencing in the path planner, and are fed as data into the task sequencing algorithm in the form of distances between tasks. Since feasible collision-free robot paths are very computationally demanding to obtain, IPS uses an iterative process where the path planner calculates only a subset of the paths by using information from the task sequence optimization algorithm. The process begins by finding a solution to the task sequencing problem given the shortest possible robot paths between tasks without considering any obstacles. The sequence is then sent to the path planner which checks how the robots should move in order to feasibly execute it, taking collisions into account. This results in new (potentially longer) distances between tasks which are sent to the task sequencing algorithm which finds a new solution etc. This process may be stopped when some minimum threshold for improvement is not met, or after a fixed number of iterations.

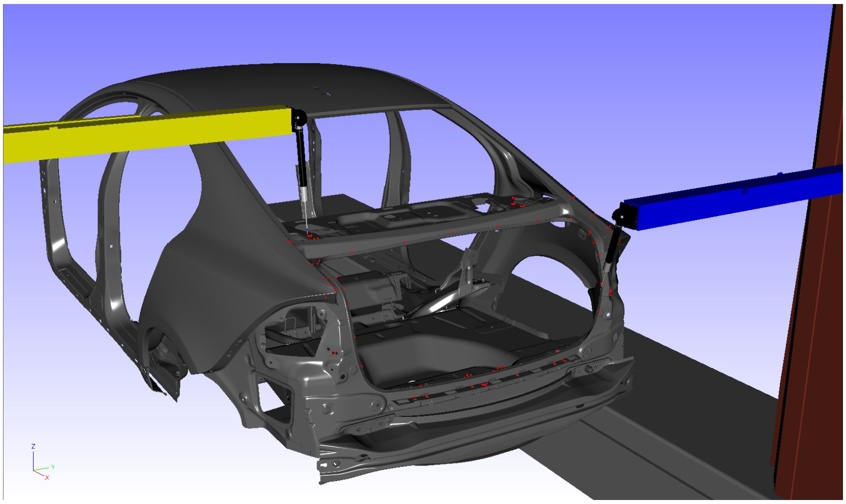


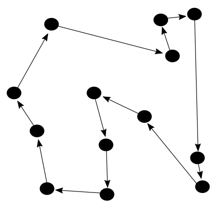
Figure 1.1: Simulation in IPS of a station with two coordinate measuring machines working on a car body.

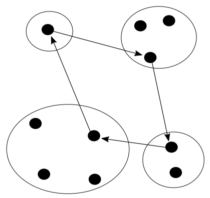
The distances between tasks may become several times as long after finding feasible robot paths, and generally the path planning affects the cycle time much more than the task sequencing. Additionally, optimizing the sequence of tasks with an incomplete set of properly planned paths is in itself an approximation of the optimal solution. Therefore, spending large amounts of computational power on finding an optimal task sequence in each iteration is normally not desired, and heuristic algorithms are utilized more often. However, exact optimization methods may still be interesting as solvers if they are sufficiently fast, and as tools that can evaluate the performance of heuristics.

This thesis presents algorithms for finding approximate and optimal solutions to the problem of sequencing tasks on robot stations with one or several automated robots such that the cycle time of the whole process is minimized. The many degrees of freedom of the robots enable tasks to be performed in many different ways. Moreover, tasks may be constrained to be performed in some partial order as to ensure the integrity of the process and/or the quality of the product. This problem is modeled as a version of the famous combinatorial optimization problem known as the Traveling Salesman Problem (TSP).

Normally the TSP is defined as the task of finding the minimum cost Hamiltonian cycle (or *tour*) in an edge-weighted graph but many variations with extra constraints or complications have been studied. The asymmetric case (ATSP) allows costs between vertices to be asymmetric and is defined on a directed graph. In the Generalized TSP (GTSP) the vertex set is partitioned into a family of disjoint and non-empty subsets. The GTSP tour is then required to visit exactly one vertex in each subset such that it minimizes the costs of the edges that are traversed. The Precedence Constrained ATSP (PCATSP), and the equivalent Sequential Ordering Problem (SOP), consists of finding a directed tour such that a partial order defined by pairwise precedence relations is respected. The Multiple TSP (mTSP) requires a fixed or variable number of tours such that each vertex is visited exactly once. Variations of the mTSP include assumptions on the number of starting points for the travelers and the objective of the problem. Most often the two objectives considered are minimizing the total traveling cost or minimizing the cost of the longest tour, also known as the minmax objective. A problem closely related to the mTSP is the Vehicle Routing Problem (VRP) [48].

The problem that this thesis considers is modeled as the Precedence Constrained Generalized Multiple TSP (PCGmTSP) with a fixed number of travelers, one starting point per traveler, and with the minmax objective. The single robot (or *agent*) case will be referred to as the PCGTSP, even though it is a special case of the PCGmTSP, as the difference in problem structure is enough to warrant treating it separately. The terms PCGmTSP and multiple agent PCGTSP will also be used interchangeably.


(a) A feasible ATSP tour.


(b) A feasible GTSP tour.

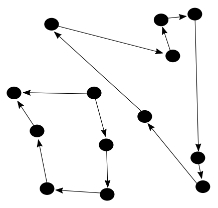

(c) A feasible mTSP solution with two travelers.

Figure 1.2: Various TSP variants.

While the PCGTSP and the PCGmTSP can be seen as aggregates of other well-studied problems, they themselves pose their own set of challenges as the methodologies for tackling the different related problems tend to clash. As an example, many effective GTSP heuristic methods rely on the fact that the edges can be exchanged very freely in any given solution, while SOP heuristics tend to exploit that the precedence constraints severely limit the edge exchange neighborhoods and the solution space in general. However, because the PCGTSP and the PCGmTSP incorporate the same or similar characteristics as the GTSP, the SOP/PCATSP, the mTSP, and even the ATSP, it is still crucial to understand how these related problems have been solved. To that end, the next section provides an overview of asymmetric TSP variants related to the PCGTSP and the PCGmTSP, and known methods for solving them.

1.2 A Review of Methods for TSP Variants

Algorithms with the purpose of solving optimization problems, in particular the TSP and variants thereof, can be mainly classified as either *exact* or *heuristic*. An exact algorithm is guaranteed to find an optimal solution or a solution within some error of the optimal solution, while a heuristic algorithm gives no guarantee on how good the produced solution is. However, for a hard problem such as the TSP, using exact algorithms for solving particularly large problem instances can easily become computationally impractical. One says that a problem instance is solvable by an exact algorithm if the algorithm finds an optimal solution and terminates within some reasonable time frame, usually 24 or 48 hours.

One of the most used types of exact algorithms for TSPs, and combinatorial optimization problems in general, is the branch-and-bound algorithm. For TSPs, the bounding procedure is commonly one of three types: relaxing integer constraints on variables in an integer linear programming (ILP) formulation and then solving the resulting linear programming (LP) problem, relaxing the vertex outdegree constraints and solving the resulting minimum spanning arborescence problem (MSAP), or relaxing the subtour elimination constraints and solving the resulting assignment problem (AP). Bounding methods based on the MSAP or the AP often involve Lagrangian relaxation coupled with some subgradient method.

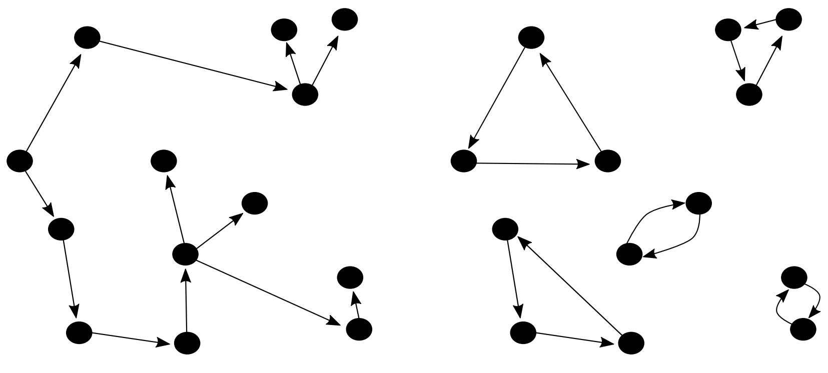

(a) A spanning arborescence. (b) A cycle cover.

Figure 1.3: Feasible solutions to the MSAP and AP in a graph.

Branch-and-bound algorithms based on LP relaxations are often augmented with some special purpose cutting plane procedure which adds additional valid constraints to the subproblems in order to tighten the lower bound produced by solving the LP problem. These so called branch-and-cut algorithms can be very powerful but require an extensive investigation of the feasible region of the problem at hand [7, 19, 20, 43]. Branch-and-cut algorithms have been proposed for the ATSP, SOP, the symmetric GTSP, and the mTSP in [22], [3], [21], and [8] respectively.

Spanning tree based bounds were first proposed for the TSP in [30, 31] and later evaluated for the asymmetric case and compared to AP based bounds (see [6]) in [49]. In the case of the GTSP and the PCATSP, the additional constraints complicate the resulting problems when relaxing the vertex outdegree constraints or the subtour eliminations constraints. For the GTSP it has been shown in [38] and [28] that the MSAP and the AP defined on a partitioned vertex set are NP-hard problems. For the PCATSP and the SOP the precedence constraints severely complicate the problem definitions. However, AP based bounds have still been considered for the GTSP in [39], where the GTSP is appropriately relaxed in order to obtain a regular AP. In [18] and [44], bounds based on the MSAP are considered for the SOP where the precedence constraints have been relaxed. For the mTSP, bounds based on polynomially solvable variations of the AP and the MSAP have been considered in [26], [1], and [24].

The dynamic programming approach (first proposed in [29]) to solving TSP variants is much less common but is still considered as a viable option. It has been shown that the PCATSP is solvable in linear time given a special structure on the precedence constraints [5]. Because of the extreme memory requirement of the dynamic programming algorithm, normally only smaller problem instances are solvable. However, so called state space relaxation schemes which limit the state space of the dynamic programming algorithm have been proposed for both the ATSP and the SOP [9, 14, 15, 35]. The resulting bounds can then be utilized within some branch-and-bound framework.

Since the TSP is such a hard problem to solve, there is a long tradition of developing effective heuristic algorithms. Local search heuristics are a type of heuristic which take a feasible solution and attempt to improve it through some problem specific manipulation. For the TSP, one of the most widely used types of local search heuristics are edge exchange based heuristics, also known as k-opt and k-exchange. A very popular adaptive edge exchange heuristic is the Lin-Kernighan heuristic [37] which was made more efficient in [32] and [33]. This heuristic has been generalized and applied to the GTSP in [36] and [34], and to the mTSP in [41]. An efficient and more restricted form of edge exchange heuristic has been developed for the SOP in [23].

So called metaheuristics are a very popular type of heuristic algorithms. They make very little assumptions about the optimization problem itself and can therefore be used as a higher level framework for a large class of problems. They are often stochastic and inspired by natural phenomena such as ant colonies, swarm behavior, and evolution, to name a few. Normally, the metaheuristics are utilized as a guided sampling of the solution space, and in conjunction with some local search heuristic. There is an abundance of literature on metaheuristic approaches to TSP variants. For some prominent examples see [2, 23, 27, 45–47].

Research on the PCGTSP is fairly scarce. In [12], the PCGTSP was transformed into an equivalent PCATSP and then a known SOP heuristic was utilized. A dynamic programming approach which extends the results found in [5] has been presented in [13]. A heuristic approach to the PCGTSP with some extra constraints has been considered in [16] and [17]. A metaheuristic approach was developed and compared to a deterministic edge exchange based heuristic and a generic exact solver in [42].

For the multiple agent case, seemingly only [25] has treated the PCGmTSP as defined in this thesis. However, only two agents are considered, and some extra spatial and logical constraints are imposed. In order to solve the problem a mathematical model is presented, and a heuristic based on a large neighborhood search is evaluated.

1.3 Contribution

The contributions in this thesis are centered around three algorithms. A metaheuristic approach to solving the PCGTSP (Paper I), an exact method for the PCGTSP (Paper II), and a metaheuristic for the PCGmTSP (Paper III). For the main contributions in each paper see the lists below.

Paper I:

- An Ant Colony Optimization based metaheuristic for solving the PCGTSP.
- An adaptation of the local search heuristic developed in [23].
- An introduction to an industrial process where the PCGTSP naturally arises.

Paper II:

- A first branch-and-bound based algorithm for solving the PCGTSP.
- A generalization of the history utilization pruning technique presented in [44].
- A new branching technique applicable to the GTSP where group sequences are enumerated and shortest path calculations are utilized.
- A novel assignment problem based bound for the GTSP.
- An experiment based evaluation of different bounding methods for the PCGTSP.

Paper III:

- An Ant Colony Optimization based metaheuristic for solving the PCGmTSP
- Adaptations of known local search heuristics which have been utilized in algorithms for the VRP, the SOP, and the machine scheduling problem.
- An introductory analysis of the nature of common TSP and VRP local search neighborhoods when applied to the PCGmTSP with the minmax objective.

1.4 Outline

Chapter 2 formally describes the PCGTSP, the PCGmTSP, and the principal notation for the rest of the thesis. Chapter 3 presents the algorithms for the PCGTSP. Section 3.1 describes the metaheuristic approach and Section 3.2 describes the exact branch-and-bound based algorithm. The metaheuristic approach to the PCGmTSP and the additional local search heuristics are presented in Chapter 4. Chapter 5 contains some additional results not found in the appended papers. In Chapter 6, some concluding remarks and suggestions for continuing the work of this thesis is given.

2. Problem Descriptions

2.1 Single Agent PCGTSP

Let $G = (V, E)$ be a directed edge-weighted graph with vertex set V, $|V| = n$, directed edge set $E \subseteq V \times V$, and edge costs c_{ij}. Let $V_1, \dots, V_m$ be a partition of V, i.e. a family of subsets of V such that $V_p \cap V_q = \emptyset$ for $p, q = 1, \dots, m$, $p \neq q$, and $\bigcup_{p=1}^{m} V_p = V$. Each subset in the partition is called a *group*. Let $M = \{1, \dots, m\}$ be the set of group indices and let $g(v) \in M$ be the index of the group in which the vertex v is contained. So, $v \in V_{g(v)}$ always holds. Let the directed acyclic graph $G' = (M, \Pi)$, $\Pi \subset M \times M$, define a partial ordering of the groups. This partial ordering is what defines the *precedence constraints*. These constraints dictate that if $(p, q) \in \Pi$, then V_p must be visited before V_q. Any precedence relation induced by transitivity is assumed to be included in Π. Which is to say, if $(p, q) \in \Pi$ and $(q, r) \in \Pi$, then $(p, r) \in \Pi$. Assume that V_1 is a predetermined start group.

The Precedence Constrained Generalized Traveling Salesman problem is then to find a cycle in G, hereby known as a *tour*, such that it starts at V_1, visits exactly one vertex in every group in an order which respects the precedence constraints, returns to V_1, and minimizes the total cost of all traversed edges.

2.2 Multiple Agent PCGTSP

As in the single agent case, assume a directed edge-weighted graph $G = (V, E)$ with a partitioned vertex set, $V_1, \dots, V_m$, and group index set $M = \{1, \dots, m\}$.

Now let A be a predetermined number of travelers, hereby known as *agents*, which are to share the task of visiting exactly one vertex in each group. Assume, without loss of generality, that each vertex $v \in V$ has been assigned a unique agent which is able to visit it, and denote this by $\Lambda(v)$. In other words, agents may never have access to the same vertex. However, agents will be allowed to have access to the same groups. Let $\Lambda(s, p) \in \{1, \ldots, A\}$ denote the agent which visits group V_p in a solution s.

Let $c_{ij}^{(e)}$ be the time it takes to traverse edge (i, j), and let $c_i^{(v)}$ be the time it takes to process vertex i. Furthermore, let $\tilde{c}_{ij} = c_{ij}^{(e)} + c_j^{(v)}$. Whenever vertex processing costs are present in a single agent setting one can simply add them to the incoming or outgoing edge costs and then discard them. However, in a multiple agent setting it will be necessary to separate the start and end times of every vertex visited in a solution. Let $T_{\text{sta}}(s, p)$ and $T_{\text{end}}(s, p)$ be the start and end time, respectively, of the processing of the group V_p in a solution s.

For each agent $a = 1, \ldots, A$, assume that $g_0(a)$ is the index of a predetermined start group for the tour of agent a. So for all $a = 1, \ldots, A$, and any feasible solution s it must hold that $\Lambda(s, g_0(a)) = a$ and $T_{\text{sta}}(s, g_0(a)) = 0$.

As before, define the precedence constraints by a directed acyclic graph $G' = (M, \Pi)$. However, since the PCGmTSP requires A disjoint tours, the interpretation of what it means to fulfill the precedence constraints needs to be revised. A natural interpretation of precedence constraint fulfillment is to require that $T_{\text{end}}(s, p) \leq T_{\text{sta}}(s, q)$ for every $(p, q) \in \Pi$. For precedence constraints with $(p, q) \in \Pi$ and $\Lambda(s, p) = \Lambda(s, q)$, so-called *intra agent constraints*, this is a correct and equivalent interpretation as in the single agent case. But when one has $(p, q) \in \Pi$ and $\Lambda(s, p) \neq \Lambda(s, q)$, so-called *inter agent constraints*, labeling any solution where $T_{\text{end}}(s, p) > T_{\text{sta}}(s, q)$ as infeasible is unnecessary. Instead, assume that whenever such a scenario occurs, a delay is added to $T_{\text{sta}}(s, q)$ such that $T_{\text{end}}(s, p) = T_{\text{sta}}(s, q)$.

The Precedence Constrained Generalized Multiple Traveling Salesman problem is then to find A disjoint tours which respect the precedence constraints according to the definition above, and cover each group exactly once such that the length of the longest tour is minimized.

2.2.1 Disjunctive Graph

In order to better understand how inter agent constraints affect a PCGmTSP solution, in particular if and how delays are incurred, a disjunctive graph representation of the solution will be utilized.

A disjunctive graph representation is depicted in Figure 2.1. It consists of all agent tours represented as paths, and directed edges with zero processing time for every inter agent constraint. All edges which represent the sequencing within the agents' tours are called *conjunctive*, while the edges which represent the inter agent constraints are called *disjunctive*. In Figure 2.1, the disjunctive edge from vertex 3 to vertex 6 means that 3 must precede 6.

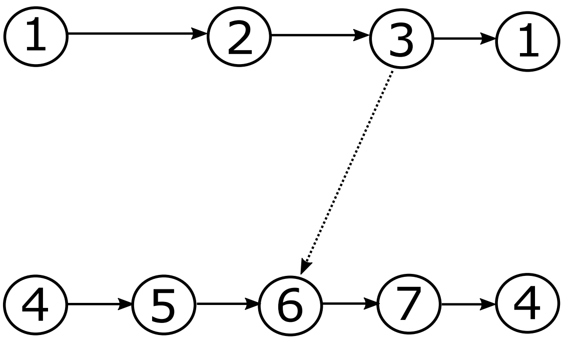


Figure 2.1: Disjunctive graph representation of a two agent solution. Disjunctive edges are dotted.

Any feasible PCGmTSP solution must have an acyclic disjunctive graph representation. Otherwise the order imposed by the agents' tours is incompatible with the partial ordering enforced by the precedence constraints. Such a *cyclic* solution is illustrated in Figure 2.2. This will be particularly important to keep in mind when developing local search heuristics as known means of manipulating and improving feasible TSP solutions, such as edge exchange heuristics, may easily lead to cyclic solutions if one is not careful.

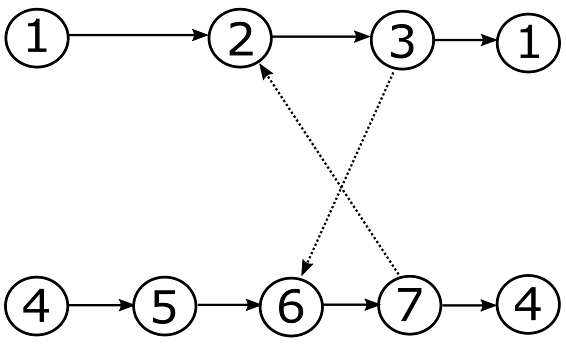


Figure 2.2: Disjunctive graph representation of an infeasible solution. Disjunctive edges are dotted.

A known result [4] is that the cost of the longest path to any vertex $i \in V$ in the disjunctive graph representation of a solution s is equal to $T_{\text{sta}}(s, g(i))$, with any eventual delays included. Furthermore, the algorithm for computing the longest path includes a topological sorting step which detects if any cycles occur within the graph. The processing times of the vertices may be used to determine the time margin of each inter agent constraint, which is the amount of time the groups involved in the constraint can be shifted before incurring a delay. The time margins can then in turn be used to estimate the delays incurred by manipulating a given solution without recomputing the longest path.

The longest path algorithm will be utilized to fully analyze a PCGmTSP solution by checking for cycles, incurred delays, computing the time margins of inter agent constraints, and determining the cycle time and time length of the individual agents' tours. However, because of its relatively expensive computational time, it will be used conservatively and simpler estimations will be employed within the heuristic algorithms.

3. Approximating and Solving the Single Agent PCGTSP

This chapter describes a metaheuristic approach based on the Hybridized Ant Colony System (HACS) algorithm [23], and a novel branch-and-bound algorithm for the PCGTSP.

3.1 Hybridized Ant Colony System (HACS)

The HACS metaheuristic is a non-deterministic algorithm which is inspired by the behavior of ants. The overarching idea is to model K generations of P ants, each of which generate paths in a probabilistic manner in a given graph. A generation consists of letting every ant generate one path each, and the ant which has produced the "best" path so far deposits pheromones along the edges of its path. This makes these edges more attractive to traverse for ants in the following generations.

Let $\tau_{ij} \in [0, 1]$ be the pheromone deposited along edge $(i, j) \in E$ and let $\eta_{ij} = 1/c_{ij}$ be a fixed visibility parameter which gives a fixed measurement of how attractive an edge is. The pheromone deposits contribute to the exploitation of good solutions. However, in order to avoid stagnation, a pheromone dissipation parameter, $\rho \in [0, 1]$, is introduced. When an edge $(i, j) \in E$ is traversed by an ant, the corresponding pheromone deposit is updated according to the local rule:

$$\tau_{ij} = (1-\rho)\tau_{ij} + \rho\tau_0 \tag{3.1}$$

where τ_0 is the initial pheromone level of all edges. Moreover, after each generation the pheromone deposits are updated according to the global rule:

$$\tau_{ij} = (1-\rho)\tau_{ij} + \rho/C(\bar{s}) \tag{3.2}$$

for every $(i,j) \in \bar{s}$, where $\bar{s}$ is the best solution found so far and $C(\bar{s})$ is its total cost.

In each step of the path generation algorithm, an ant must choose which edge to traverse given that the ant has generated a path leading up to vertex $i \in V$. Assume that $\alpha, \beta \geq 1$ are parameters which regulate the relative importance of the pheromone deposit and the visibility parameter, respectively, when choosing an edge. Let ψ_{ij} denote the attractiveness of edge $(i,j) \in E$ and let it be computed as:

$$\psi_{ij} = (\tau_{ij})^{\alpha}(\eta_{ij})^{\beta}. \tag{3.3}$$

Furthermore, assume that $V(\tilde{s})$ is the set of vertices which are allowed to be visited next given the partial solution $\tilde{s}$ that ends in vertex i. The ant then chooses to traverse edge $(i,j^*) \in E$, $j^* \in V(\tilde{s})$, such that

$$\psi_{ij^*} \geq \psi_{ij} \qquad\qquad \forall (i,j) \in E : j \in V(\tilde{s}), \tag{3.4}$$

with probability d_0. This is the deterministic rule which chooses the edge which is the most attractive. The probabilistic rule is chosen with probability $(1-d_0)$, and dictates that the ant chooses edge $(i,j) \in E$ with probability

$$f_{ij} = \begin{cases} \dfrac{\psi_{ij}}{\sum\limits_{l \in V(\tilde{s})} \psi_{il}}, & \text{if } j \in V(\tilde{s}) \\ 0, & \text{otherwise.} \end{cases} \tag{3.5}$$

Moreover, for each path that is generated a 3-opt local search heuristic and a vertex optimization procedure is applied. These algorithms are described in

Sections 3.1.1 and 3.1.2. The HACS framework and the path generation algorithm are outlined below in Algorithm 3.1 and 3.2, respectively.

Algorithm 3.1 HACS framework

```
1. Set $k := 1$ and set $\bar{s} = \mathbf{0}^m$ with $C(\bar{s}) = \infty$.
2. Set $p := 1$
3. Generate a solution $s_p^k$ according to solution generation algorithm, and apply local
   search heuristics. If $C(s_p^k) < C(\bar{s})$ then set $\bar{s} = s_p^k$.
4. If $p < P$ then set $p := p + 1$ and go to step 3.
5. Set $k := k + 1$. Take the best solution found so far, $\bar{s}$, and update the pheromone
   levels as $\tau_{ij} = (1 - \rho)\tau_{ij} + \rho / C(\bar{s})$ for every $(i, j) \in \bar{s}$.
6. If $k < K$ then go to step 2. Otherwise return overall best solution that was found
   and stop.
```

Algorithm 3.2 Path generation for HACS

```
1. Initialize the solution $\tilde{s}$ by setting the first vertex to the predetermined start vertex
   and set $k := 2$.
2. Compute the set of vertices allowed to be sequenced next in the solution, $V(\tilde{s})$,
   by taking into account the precedence constraints and the groups already visited
   in $\tilde{s}$.
3. Let $d \in [0, 1]$ be a uniformly distributed random number. If $d \leq d_0$ then choose
   the edge $(i, j)$ according to the deterministic rule in Equation 3.4, i.e. the edge
   which is feasible and has the highest probability is chosen. If $d > d_0$ choose to
   traverse the edge $(i, j)$ according to probabilistic rule in Equation 3.5.
4. If edge $(i, j)$ is traversed then update the pheromone deposits according to $\tau_{ij} :=$
   $(1 - \rho)\tau_{ij} + \rho\tau_0$ and add $(i, j)$ to $\tilde{s}$.
5. If $k < m$ set $k := k + 1$ and go to step 2. Otherwise return $\tilde{s}$.
```

3.1.1 Vertex Selection

Choosing an optimal vertex selection given a sequence of groups can be formulated as a simple shortest path problem in a layered network [21]. Assume a given feasible tour represented as a straight path of groups, i.e.

$$\sigma = (V_{p_1}, \ldots, V_{p_m}, V_{p_{m+1}}) \tag{3.6}$$

with $V_{p_1} = V_{p_{m+1}} = V_1$. Now take the shortest of the $|V_1|$ shortest paths which run through the group sequence σ represented as a layered network where each group's vertices consists of one layer (see Figure 3.1).

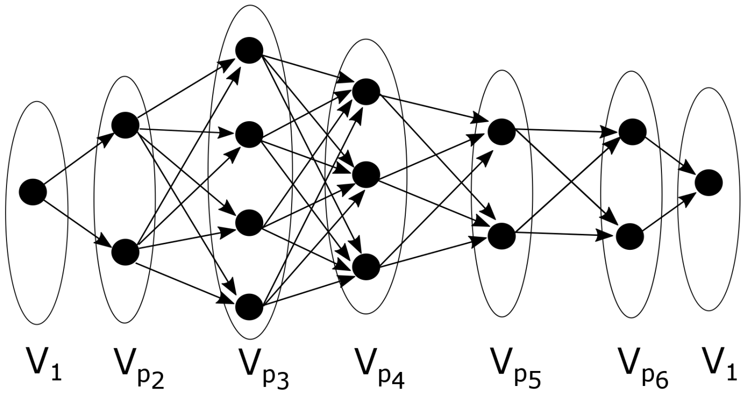


Figure 3.1: A layered network representation of a feasible group sequence. Shortest path through the network gives an optimal vertex selection.

There are known algorithms with polynomial worst case time complexity which are able to solve the shortest path problem in a directed acyclic graph. This makes vertex selection a relatively simple problem compared to group sequencing. However, applying a full optimization of the vertex selection every time a solution is modified is still quite cumbersome. Therefore, full vertex selection optimization is only applied after a solution has been constructed and after a local search heuristic is not able to improve the solution any further.

3.1.2 Path Preserving 3-opt

The heuristic which is applied after a feasible solution is generated in the HACS algorithm (step 3 in Algorithm 3.1), is the path preserving 3-opt (PP3opt) developed for the SOP in [23].

Normally, a 3-opt heuristic takes a feasible tour, and attempts to remove 3 edges and add 3 edges, an operation known as a 3-exchange, such that feasibility is retained and the cost of the tour is reduced. For the symmetric TSP, one can check both feasibility and eventual improvement in constant time for each candidate 3-exchange. However, verifying precedence constraints and checking for improvement with asymmetric costs increases this to $O(n)$. The idea behind the PP3opt heuristic is simple yet powerful. Since precedence constraints and the asymmetric costs severely hamper the ability of traditional k-opt heuristics to efficiently search most of the space of feasible exchanges, the PP3opt heuristic limits the search to 3-exchanges which are efficiently verifiable, namely 3-exchanges which preserve the orientation of the solution (see Figure 3.2). This restriction of the search neighborhood together with the utilization of a special labeling procedure for fast verification of the precedence constraints, enables the PP3opt to retain the same worst case time complexity as a regular 3-opt applied to the symmetric TSP.

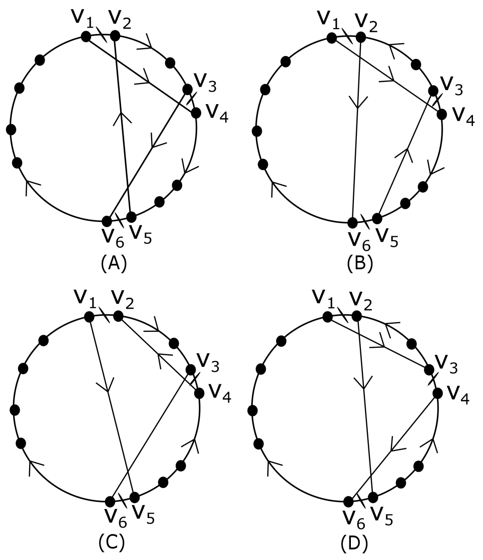


Figure 3.2: The four different types of 3-exchange when edges (v_1, v_2), (v_3, v_4), and (v_5, v_6) are removed. (A) is path preserving.

When applied to the PCGTSP, small modifications are done in order to improve vertex selection. If one ignores vertex selection completely, the improvement which is computed for each 3-exchange in the PP3opt is a misrepresentation of what the actual improvement is. Conversely, checking every possible vertex selection for each candidate 3-exchange is very computationally expensive. In order to achieve a middle ground, a local vertex improvement procedure is introduced. Assume that the edges (v_1, v_2), (v_3, v_4), and (v_5, v_6) are removed, and some combination of edges which reconnect these vertices is added. Then for each group $g(v_i)$, $i = 1, \dots, 6$, a new vertex is chosen such that the cost of the tour is reduced. This is done sequentially in the order in which the groups are visited in the tour without reconsidering vertex selections of previous groups. The local vertex improvement procedure is applied after every 3-exchange. When no more improving and feasible 3-exchanges can be found, a full optimization of the vertex selection is performed.

3.2 Branch-and-Bound Algorithm

The branch-and-bound algorithm proposed in this thesis utilizes a novel branching strategy where vertex selection is never fixed. Instead of branching on the vertices, which would correspond to enumerating all feasible tours, the algorithm branches on the order of groups. In other words, the search tree starts with a root node where only the starting group is fixed, and every branch corresponds to which group is to be visited next. The idea is that since the number of feasible group sequences must be less than or equal to the number of feasible tours, this strategy will limit branching.

Subproblems are formulated by utilizing shortest path calculations for their corresponding group sequence. Assume that a node in the branch-and-bound search tree attempting to solve the PCGTSP instance $\mathcal{P}$ corresponds to the fixed group sequence $\sigma = (V_1, \dots, V_{p_r})$. The corresponding subproblem $\mathcal{P}(\sigma)$ is then the PCGTSP instance $\mathcal{P}$ but constrained to find a tour which begins with the group order defined by σ. A problem equivalent to $\mathcal{P}(\sigma)$ can be formulated as following:

Definition 3.1. *Given a PCGTSP instance $\mathcal{P}$ and a feasible group sequence $\sigma = (V_1, \dots, V_{p_r})$, define $\mathcal{P}_1(\sigma)$ as a PCGTSP instance with*

- *the same groups, vertices and edges as in $\mathcal{P}$, with the exception of $V_{p_2}, \ldots, V_{p_{r-1}}$ and all their associated vertices and edges,*
- *the same precedence constraints as in $\mathcal{P}$ (taking into account the removed groups),*
- *all outgoing edges from V_1 and all incoming edges to V_{p_r} removed except for $(i, j) \in E : i \in V_1, j \in V_{p_r}$,*
- *and all edge costs c_{ij} for the edges $(i, j) \in E : i \in V_1, j \in V_{p_r}$, replaced by the cost of the shortest path from $i \in V_1$ to $j \in V_{p_r}$ when traversing the groups $V_{p_2}, \ldots, V_{p_{r-1}}$ in the order defined by σ.*

However, by separating the partial group sequence defined by σ from the rest of the instance one can formulate a powerful pruning technique which utilizes information from already processed tree nodes. This will be explained further in Section 3.2.2. The problem of finding a feasible tour (with respect to $\mathcal{P}(\sigma)$) which only takes into account the edge costs of the path from V_{p_r} to V_1 may be defined as:

Definition 3.2. *Given a PCGTSP instance $\mathcal{P}$ and a feasible group sequence $\sigma = (V_1, \ldots, V_{p_r})$, define $\mathcal{P}_2(\sigma)$ as a PCGTSP instance with*

- *the same groups, vertices and edges as in $\mathcal{P}$, with the exception of $V_{p_2}, \ldots, V_{p_{r-1}}$ and all their associated vertices and edges,*
- *the same precedence constraints as in $\mathcal{P}$ (taking into account the removed groups),*
- *all outgoing edges from V_1 and all incoming edges to V_{p_r} removed except for $(i, j) \in E : i \in V_1, j \in V_{p_r}$,*
- *and all edge costs c_{ij} for the edges $(i, j) \in E : i \in V_1, j \in V_{p_r}$ set to zero.*

Assume that $z^*(\mathcal{P})$ is the optimal tour cost of a problem instance $\mathcal{P}$ and that $c_{\text{min}}(\sigma)$ is the cost of the shortest path from V_1 to V_{p_r} when traversing the groups $V_{p_2}, \ldots, V_{p_{r-1}}$ in the order defined by σ. Then the following result holds:

Proposition 3.1. $z^*(\mathcal{P}(\sigma)) \geq z^*(\mathcal{P}_2(\sigma)) + c_{min}(\sigma)$

Proof. The left hand side is the cost of a tour which begins with the sequence of groups $\sigma = (V_1, \ldots, V_{p_r})$ and then traverses the rest of the groups of the problem instance $\mathcal{P}$. The left hand side is the cost of the optimal solution to the same problem as $\mathcal{P}(\sigma)$ but with the vertex selection at V_1 and V_{p_r} relaxed. That is to say, one allows the tour to enter and exit at different vertices in V_1 and V_{p_r}. This can only reduce the cost of the tour. □

Proposition 3.1 means that one can compute a valid lower bound for the problem $\mathcal{P}(\sigma)$ by first computing a lower bound for $\mathcal{P}_2(\sigma)$ and then adding the cost of the shortest path which traverses σ.

The branch-and-bound tree is traversed by using a depth-first search in order to conserve memory and to rapidly obtain complete solutions. Branching is prioritized according to the rule

$$\arg\max_{p \in M} |\{q \in M : (p, q) \in \Pi\}|. \tag{3.7}$$

The motivation for using this priority is two-fold. Firstly, groups which are required to precede many of the other groups should have a higher probability of occurring early in the optimal tour. Secondly, since the precedence constraints are almost completely relaxed in the bounding methods which are considered in this thesis, eliminating them may strengthen the lower bounds.

3.2.1 Bounding Methods

The bounding methods considered in this thesis are based on the minimum spanning arborescence problem (MSAP) and the assignment problem (AP). The MSAP is obtained by taking an ATSP and relaxing the vertex out-degree constraints which dictate that each vertex should have exactly one outgoing edge connected to it. The resulting problem then becomes to find the minimum cost directed spanning tree (arborescence) such that each vertex has exactly one incoming edge. The AP can be formulated by taking an ATSP and relaxing the subtour elimination constraints which ensure that only one cycle occurs in the solution (the tour itself). The result is the vertex disjoint cycle cover problem which can be equivalently formulated as an AP. However, due to the precedence constraints and the partitioned graph, the same relaxations applied to the PCGTSP result in intractable NP-hard problems [28, 38].

Since it is difficult to even define what a predecessor is in an arborescence or a cycle cover, the precedence constraints are almost completely relaxed. The *weak version* of a PCGTSP instance $\mathcal{P}$ is the GTSP without precedence constraints which is defined on the same graph as $\mathcal{P}$ but with all edges $(i, j) \in E : (g(j), g(i)) \in \Pi$ removed. The weak version of $\mathcal{P}$ is a relaxation of $\mathcal{P}$ and therefore its optimal tour cost must be lower than that of $\mathcal{P}$.

In order to obtain an unpartitioned graph, two different approaches are considered. The first one involves applying the Noon-Bean transformation [40] to the weak version of a PCGTSP instance. The transformation takes an asymmetric GTSP instance and defines an equivalent ATSP with n vertices. This is achieved by defining a directed cycle within each group where the edges that the cycle consists of have zero cost. Furthermore, if a group V_p with $|V_p| = r$ is given a cycle with the order $(v_1, \ldots, v_r)$ then the costs of all outgoing edges from V_p are redefined as:

$$\begin{cases} c_{v_i j}^{(\text{NB})} = c_{v_{i+1} j} & \forall j \notin V_p, i = 1, \ldots, r-1, \\ \\ c_{v_r j}^{(\text{NB})} = c_{v_1 j} & \forall j \notin V_p. \end{cases} \tag{3.8}$$

In order to ensure that the edges within the cycle are the only zero cost edges in the graph, edges not belonging to the cycle have their corresponding cost offset by a value $c_{\text{off}}^{(\text{NB})}$. In other words, for every edge $(i, j) \in E$ such that $(i, j) \neq (v_i, v_{i+1})$, $i = 1, \ldots, r-1$, and $(i, j) \neq (v_r, v_1)$, the corresponding edge costs are defined as:

$$c_{ij}^{(\text{NB})} = c_{ij} + c_{\text{off}}^{(\text{NB})}. \tag{3.9}$$

By using these edge costs and removing the groups (but retaining the set of all vertices) one may define the following ATSP instance:

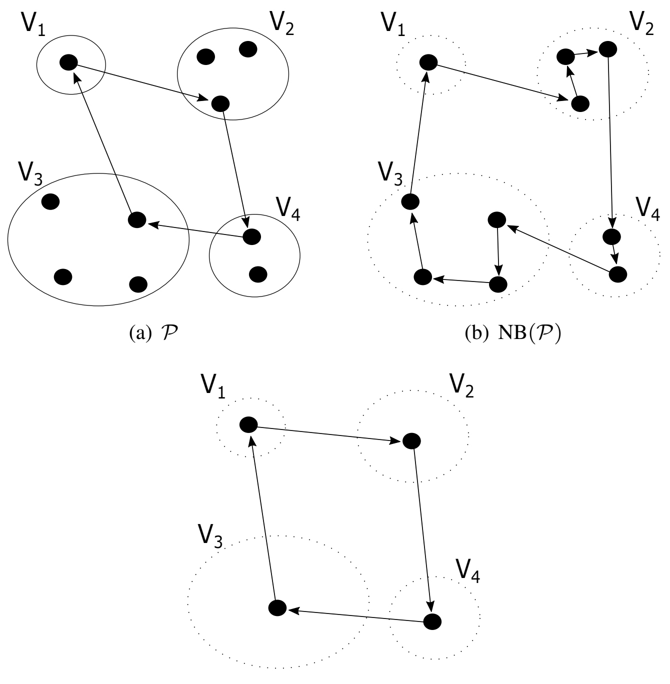
Definition 3.3. *For any PCGTSP instance $\mathcal{P}$, let* $\text{NB}(\mathcal{P})$ *denote the ATSP instance which arises when applying the Noon-Bean transformation [40] to the weak version of $\mathcal{P}$.*

The second approach involves relaxing the vertex choice constraints which enforce the rule that one must enter and exit the same vertex when visiting a group. It has been shown in [39] that by doing this one may define each group as a single vertex and all edge costs as:

$$c_{pq}^{(\mathrm{NC})} = \min_{(i,j)\in V_p \times V_q} c_{ij} \qquad \forall (p,q) \in M \times M : (q,p) \notin \Pi. \tag{3.10}$$

In other words, the following ATSP instance with m vertices is formulated:

Definition 3.4. *For any weak version of a PCGTSP instance $\mathcal{P}$, let* $\mathrm{NC}(\mathcal{P})$ *be the ATSP instance defined on the graph $G_{\mathrm{NC}} = (M, E_{\mathrm{NC}})$ where $E_{\mathrm{NC}} = \{(p,q) \in M \times M : (q,p) \notin \Pi\}$. For every $(p,q) \in E_{\mathrm{NC}}$ the edge costs are defined as in Equation 3.10.*



(a) $\mathcal{P}$ (b) $\mathrm{NB}(\mathcal{P})$

(c) $\mathrm{NC}(\mathcal{P})$

Figure 3.3: Feasible tours in a GTSP instance $\mathcal{P}$ and the transformations $\mathrm{NB}(\mathcal{P})$ and $\mathrm{NC}(\mathcal{P})$.

A couple of things should be noted about these transformed problems. If applied to the same PCGTSP instance $\mathcal{P}$, the problem $\text{NC}(\mathcal{P})$ is always smaller than or equal to the size of $\text{NB}(\mathcal{P})$ in terms of the size of the graphs which these problems are defined on (since $m \leq n$). However, if $z^*(\cdot)$ is the optimal tour cost of a problem instance, then the following relations between the problem instances hold (given that $z^*(\text{NB}(\mathcal{P}))$ has been appropriately adjusted for the edge cost offsets $c_{\text{off}}^{(\text{NB})}$):

$$z^*(\mathcal{P}) = z^*(\text{NB}(\mathcal{P})) \geq z^*(\text{NC}(\mathcal{P})). \tag{3.11}$$

So while $\text{NC}(\mathcal{P})$ is defined on a smaller graph than $\text{NB}(\mathcal{P})$, it is a potentially weaker formulation. Furthermore, $\text{NB}(\mathcal{P})$ is a problem which is particularly ill-suited for the AP bound since any group V_p with $|V_p| > 1$ defines a zero cost cycle in the graph. Consider a PCGTSP instance $\mathcal{P}$ with $|V_p| > 1$, $\forall p \in M$. Then the optimal cycle cover in the graph of $\text{NB}(\mathcal{P})$ has zero cost which is a trivial bound.

3.2.1.1 An Alternative Assignment Problem Bound

One can utilize a more general version of the problem $\text{NC}(\mathcal{P})$ when computing the AP bound by replacing the edge costs with so called L-paths:

Definition 3.5. *For a PCGTSP instance $\mathcal{P}$ and for a fixed integer $L \geq 1$, an L-path is a feasible path (a path which could be a part of a feasible tour in $\mathcal{P}$) consisting of exactly L edges 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. Let the problem $\text{NC}_L(\mathcal{P})$ be the same as $\text{NC}(\mathcal{P})$ but with edge costs $c_{pq}^{(\text{NC})} = d_L(p,q)$.*

Note that $d_1(p,q)$ is simply the minimum cost of all edges from p to q and the resulting problem $\text{NC}_1(\mathcal{P})$ is equal to $\text{NC}(\mathcal{P})$. For every fixed $L > 1$ computing the L-path distances $d_L(p,q)$ can be done in polynomial time by using a dynamic programming method. One can use the following result in order to compute a bound for $\text{NC}_L(\mathcal{P})$:

Proposition 3.2. *The minimum cost of a cycle cover in $\text{NC}_L(\mathcal{P})$ divided by L, is a lower bound on the optimal tour cost of $\mathcal{P}$.*

Proof. See Paper II. □

3.2.2 History Utilization

By using the subproblem definition $\mathcal{P}_2(\sigma)$ described in Definition 3.2, the history utilization pruning technique presented in [44] for the SOP can be generalized for the PCGTSP. In order to adequately describe the technique the following equivalence relation among branch-and-bound tree nodes is useful:

Definition 3.6. *For every pair (S, r), $S \subseteq M$, $|S| > 1$, and $r \in S$ such that $(r, q) \notin \Pi$ when $q \in S$, define $\mathcal{T}(S, r)$ 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 tree nodes belonging to the same set $\mathcal{T}(S, r)$ are said to be equivalent.*

In the remainder of this section, consider an unprocessed tree node $\mathcal{N}(\sigma) \in \mathcal{T}(S, r)$ with partial group sequence $\sigma = (V_1, \ldots, V_r)$. Let $P_{ij}^{(\sigma)}$ be the shortest path which leads from $i \in V_1$ to $j \in V_r$ through σ, and let $c(P_{ij}^{(\sigma)})$ denote its cost. 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 $\mathcal{T}(S, r)$ has been processed then $c(P_{ij}^{(S,r)}) = \infty$. The following result enables the pruning technique:

Proposition 3.3. *If $c(P_{ij}^{(S,r)}) < c(P_{ij}^{(\sigma)}), \forall (i, j) \in E; i \in V_1, j \in V_r$, then there can't exist a solution to the PCGTSP which includes σ and has smaller total cost than a solution which includes $P_{ij}^{(S,r)}$, $(i, j) \in E : i \in V_1, j \in V_r$. In other words, the tree node $\mathcal{N}(\sigma)$ can be pruned.*

Proof. See Paper II. □

If $c(P_{ij}^{(\sigma)}) < c(P_{ij}^{(S,r)})$ for some $(i, j) \in E : i \in V_1, j \in V_r$ then the tree node $\mathcal{N}(\sigma) \in \mathcal{T}(S, r)$ cannot be pruned according to Proposition 3.3. However, if another tree node $\mathcal{N}(\tilde{\sigma}) \in \mathcal{T}(S, r)$ has been processed before $\mathcal{N}(\sigma)$, and the lower bound $z_{\text{LB}}(\mathcal{P}_2(\tilde{\sigma}))$ on the corresponding problem $\mathcal{P}_2(\tilde{\sigma})$ has been stored, then one can obtain a lower bound for $\mathcal{P}(\sigma)$ directly by computing:

$$z_{\text{LB}}(\mathcal{P}(\sigma)) = z_{\text{LB}}(\mathcal{P}_2(\tilde{\sigma})) + \min_{(i,j) \in V_1 \times V_r} c(P_{ij}^{(\sigma)}). \tag{3.12}$$

This follows from Proposition 3.1 and the fact that $\mathcal{P}_2(\sigma) = \mathcal{P}_2(\tilde{\sigma})$.

4. Approximating the Multiple Agent PCGTSP

4.1 Hybridized Ant Colony System (HACS)

The HACS for the PCGmTSP largely follows the same procedure as for the PCGTSP outlined in Section 3.1. Since every vertex edge $(i, j) \in E$ is uniquely associated with an agent, there is no need to specify the agent when an ant chooses to traverse an edge. Therefore, the same procedures and quantities as for the single agent case can be used.

However, it should be noted how $V(\tilde{s})$, the set of vertices which are feasible to visit next given a partial solution $\tilde{s}$, is defined. Assume that each of the A partial tours in $\tilde{s}$ end at the vertices $i_1, \ldots, i_A$. Then a vertex j is in $V(\tilde{s})$ if the following conditions hold:

- $V_{g(j)}$ has not been visited in $\tilde{s}$.
- $\Lambda(j) = \Lambda(i_a)$ and $(i_a, j) \in E$ for some $a = 1, \ldots, A$.
- All groups V_p such that $(p, g(j)) \in \Pi$ have been visited in $\tilde{s}$.

The last condition might seem unnecessarily severe. After all, one may formulate a rule where an agent is allowed to visit a group V_q as long as there is at least one other agent which is able to visit all unvisited groups $V_p : (p, q) \in \Pi$. This rule might generate solutions with many delays but enables the HACS algorithm to search a more diverse set of solutions. However, it turns out that such a rule often leads to cyclic solutions, and becomes harder and harder to verify as the path generation algorithm progresses.

4.2 Local Search Procedure

For the single agent PCGTSP only one local search heuristic, PP3opt, is applied to a single tour, and therefore the local search procedure could be formulated as a simple descent search. For the multiple agent case, there are several challenges which motivate a revised local search procedure, such as several local search heuristics with possible interactions, and the expensive cycle time computation.

The different heuristics which are applied are: a slightly revised version of PP3opt, String Move which tries to move a part of an agent's tour to another agent, and Delay Removal which attempts to eliminate delays from a solution by moving groups that are involved in a delay forwards or backwards in their respective tour. Details around their implementations will be described in the coming sections.

The first thing which is modified in the local search procedure is the improvement criterion for the different heuristics. Normally, an improvement is defined with respect to a problem's objective, but using a reduction in cycle time as a measure of improvement may lead to undesired deadlocks. Assume that s_{old} is a solution given to a local search heuristic and that s_{new} is the same solution after being manipulated. Let $\boldsymbol{T}(s) = (T_1(s), \dots T_A(s))$ be a vector of the A individual agents' tour lengths of a solution s, sorted in ascending order ($T_1(s)$ is the shortest and $T_A(s)$ the longest). One may then define the following improvement measure:

$$I(s_{\text{old}}, s_{\text{new}}) = L_A(\boldsymbol{T}(s_{\text{old}}), \boldsymbol{T}(s_{\text{new}})) \tag{4.1}$$

where for all $a = 1, \dots, A$, and $\boldsymbol{x} = (x_1, \dots, x_A)$, $\boldsymbol{y} = (y_1, \dots, y_A)$, the recursive function $L_a : \mathbb{R}^A \times \mathbb{R}^A \to \mathbb{R}$ is defined as

$$L_a(\boldsymbol{x}, \boldsymbol{y}) = \begin{cases} L_{a-1}(\boldsymbol{x}, \boldsymbol{y}), & \text{if } x_a = y_a \text{ and } a > 1 \\ x_a - y_a, & \text{otherwise.} \end{cases} \tag{4.2}$$

The improvement measure $I(\cdot, \cdot)$ will naturally never consider an increase in cycle time as an improvement but will accept some cases of constant cycle time while reducing the length of an agent's tour as an improvement.

Because of possible interactions between the different local search heuristics, a general framework around them is implemented. Given a solution s, let $N_{\text{3opt}}(s)$, $N_{\text{SM}}(s)$, and $N_{\text{DR}}(s)$ be the neighborhoods of the PP3opt, String Move, and Delay Removal heuristics respectively. Given a feasible initial solution s_0 and a maximum number of iterations $k_{\max}$, the framework for the local search procedure can then be outlined as follows:

Algorithm 4.1 Local Search Framework

```
1. Set $k := 1$ and $p := 0$.
2. Try to find a solution $s_{\text{3opt}} \in N_{\text{3opt}}(s_k)$ such that $I(s_k, s_{\text{3opt}}) > 0$. If
   $I(s_k, s_{\text{3opt}}) > p$, set $p := I(s_k, s_{\text{3opt}})$ and $s_{k+1} := s_{\text{3opt}}$.
3. If $p = 0$, try to find a solution $s_{\text{SM}} \in N_{\text{SM}}(s_k)$ such that $I(s_k, s_{SM}) > 0$. If
   $I(s_k, s_{\text{SM}}) > p$, set $p := I(s_k, s_{\text{SM}})$ and $s_{k+1} := s_{\text{SM}}$.
4. Try to find a solution $s_{\text{DR}} \in N_{\text{DR}}(s_k)$ such that $I(s_k, s_{\text{DR}}) > 0$. If $I(s_k, s_{\text{DR}}) >$
   $p$, set $p := I(s_k, s_{\text{DR}})$ and $s_{k+1} := s_{\text{DR}}$.
5. If $k = k_{\max}$ or $p = 0$ then terminate and return $s_k$. Otherwise set $k := k + 1$,
   $p := 0$, and go to step 2.
```

Note that the String Move heuristic is only invoked if the PP3opt is not able to find an improving solution, while the Delay Removal heuristic is always executed.

Since computing the actual improvement for every solution in the local search neighborhoods is very expensive, heuristic specific estimations of the individual agents' tours will be used instead. The estimations are then used in order to cull the number of solutions which are to be fully analyzed. Given a solution $\tilde{s}$ and a max number of solutions to be evaluated $j_{\max}$, then a general procedure for finding an improving solution within a neighborhood $N(\tilde{s})$ is:

Algorithm 4.2 Neighborhood Search

```
1. For each $s \in N(\tilde{s})$, let $\hat{I}(\tilde{s}, s)$ be an estimation of $I(\tilde{s}, s)$ and add $s$ to a list which is sorted according to the estimations in descending order. So, if the list has $|N(\tilde{s})|$ elements then $\hat{I}(\tilde{s}, s_k) \geq \hat{I}(\tilde{s}, s_{k+1})$ holds for $k = 1, \dots, |N(\tilde{s})|$.
2. Set $k := 1$.
3. Compute the actual improvement $I(\tilde{s}, s_k)$ by computing the longest path in the disjunctive graph representation of $s_k$. If $I(\tilde{s}, s_k) > 0$ then terminate and return $s_k$.
4. If $k = j_{\text{max}}$, then terminate and return $\tilde{s}$. Otherwise, set $k := k + 1$ and go to step 3.
```

It is important to note that the list of solutions sorted according to the estimated improvements only give a slight indication of which solutions are good. It is entirely possible that all of the solutions in the list are non-improving or even infeasible.

4.2.1 Vertex Selection

In order to optimize vertex selection one can apply the same procedure as described in Section 3.1.1 to each individual tour in a PCGmTSP solution. The vertex selection algorithm is always applied before a solution is fully analyzed by computing the longest path in the disjunctive graph. Other, more local, vertex selection improvements are applied within the local search heuristics.

4.2.2 Path Preserving 3-opt

The PP3opt is only modified such that it searches all A tours for feasible path preserving 3-exchanges. The estimation of improvement for the neighborhood search are, as usual, based on the sums of costs of the edges exchanged but the delays directly caused by the 3-exchange are also taken into account. A path preserving 3-exchange may be visualized as two paths within the tour trading places (see Figure 4.1).

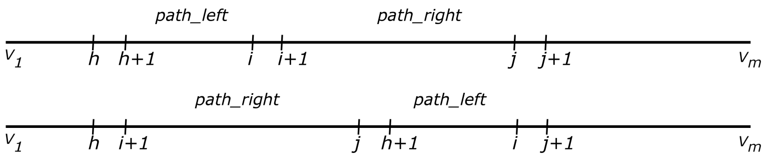


Figure 4.1: A path preserving 3-exchange for a tour visualized as a path. Top path represents the tour before the exchange, and the bottom one after the exchange.

Assume that a 3-exchange for the tour of agent a in solution s is being evaluated. If a vertex v_i in "path_left" is involved in an inter agent constraint, $(g(v_i), q) \in \Pi$ and $\Lambda(s, q) \neq a$, then a delay may be incurred in the tour of agent $\Lambda(s, q)$. Similarly, if a vertex v_i in "path_right" is involved in an inter agent constraint, $(p, g(v_i)) \in \Pi$ and $\Lambda(s, p) \neq a$, then a delay may be incurred in the tour of agent a. These types of delays are the only ones that are considered when estimating the improvement while delays caused indirectly due to chain effects are ignored.

4.2.3 String Move

The String Move heuristic is based on a known VRP heuristic [11] adapted to handle the precedence constraints and the vertex selection. It attempts to move a sequence of groups $\sigma = (V_{p_1}, \ldots, V_{p_r})$ from the tour of one agent a_f to the tour of another agent a_t. This is a path preserving operation, and therefore the same labeling procedure for fast verification of the precedence constraints used in the PP3opt heuristic can be used here.

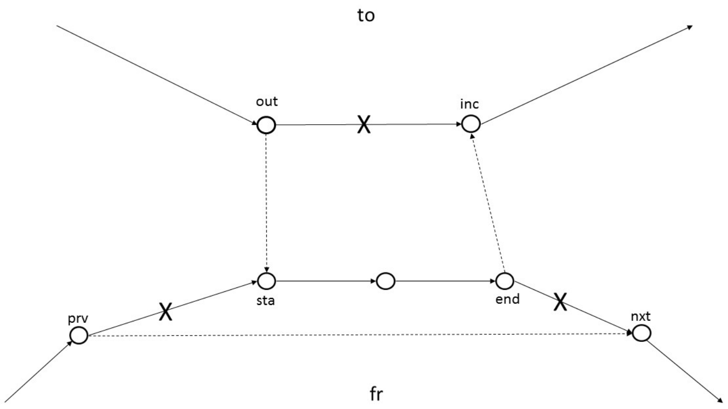


Figure 4.2: A string move. Edges with an "X" over them are removed while dashed edges are added.

Since vertices are unique to each agent, the choice of vertices within σ needs to be determined when moving it to another agent's tour. This is done in a greedy fashion. Assume that σ is removed from between the vertices v_{prv} and v_{nxt} in the tour of agent a_{f} and added to the tour of agent a_{t} between the vertices v_{out} and v_{inc}. Also assume that that $\sigma_{\text{t}} = (v_1, \ldots, v_r)$ is the sequence of vertices to be chosen to visit when σ is moved to a_{t}. So the edges (v_{prv}, v_1), (v_r, v_{nxt}) and $(v_{\text{out}}, v_{\text{inc}})$ are removed, while the edges (v_{out}, v_1), (v_r, v_{inc}), and $(v_{\text{prv}}, v_{\text{nxt}})$ are added. The String Move heuristic chooses the vertex $v_1 \in V_{p_1}$ with $\Lambda(v_1) = a_{\text{t}}$ which minimizes:

$$\lambda_1 = \tilde{c}_{v_{\text{out}} v_1} + \tilde{c}_{v_1 v_{\text{inc}}}. \tag{4.3}$$

More generally, for $i = 2, \ldots, r$, the vertex $v_i \in V_{p_i} : \Lambda(v_i) = a_{\text{t}}$ is chosen such that

$$\lambda_i = \lambda_{i-1} + \tilde{c}_{v_{i-1} v_i} + \tilde{c}_{v_i v_{\text{inc}}} - \tilde{c}_{v_{i-1} v_{\text{inc}}} \tag{4.4}$$

is minimized.

The lengths of the agents' tours are estimated by removing the cost of traversing σ in the tour of agent a_{f} and adding the cost of traversing σ_{t} in the tour of a_{t}. Furthermore, delays incurred by moving σ are also estimated and taken into account. These estimations are only based on the time shifts directly caused by moving σ.

4.2.4 Delay Removal

The Delay Removal heuristic is based on a known method for job shop scheduling problems [10]. It takes a solution s, identifies delays caused by inter agent constraints $(p, q) \in \Pi$, and tries to reduce them. This is done by attempting to move V_p backwards to an earlier place in the tour of agent $\Lambda(s, p)$ while moving V_q forwards to a later place in the tour of $\Lambda(s, q)$ such that the delay is reduced, and the solution remains feasible and is improved.

The estimation of the tour lengths takes into account the reduction in the delay, the cost of the edges which are exchanged in order to move V_p backwards and V_q forwards, and eventual new delays caused directly by the edge exchange.

5. Results

In this chapter some updated results for the HACS approach to the PCGTSP is presented. Many optimizations of the code and improved compiler settings since the results presented in Paper I has increased the speed considerably.

The problem instances "cmm00x" are derived from industrial instances of coordinate measuring machine problems. Problem instances named "020.XXX" are derived from SOP instances in the same way as outlined in Paper II. In Table 5.1 the results measured over 10 trial runs are presented. The column "Mean $\pm$ SD" shows the mean solution value and the standard deviation, and "T (s)" is the mean execution time. The optimality gap is the mean solution value (M) compared to the best known lower bound (LB) and is calculated as $(\mathrm{M}-\mathrm{LB})/\mathrm{M}$. The following parameter values are used in the HACS algorithm:

- $\alpha = 1, \beta = 2$ (attractiveness weighted towards edge length)
- $\rho = 0.1$ (evaporation parameter)
- $d_0 = 0.9$ (deterministic rule probability)
- $\tau_0 = 0.5$ (initial pheromone deposits)

Table 5.1: Updated results for the HACS heuristic.

Instance	m	n	HACS			Best known	
			Mean ± SD	Gap	T (s)	UB	LB
cmm001	12	14	49.1 ± 0.0	0.000	0.0	49.1	49.1
cmm002	15	24	20.7 ± 0.0	0.019	0.0	20.3	20.3
cmm003	17	35	20.3 ± 0.1	0.015	0.1	20.0	20.0
cmm004	90	215	47.5 ± 0.7	0.516	2.5	46.1	23.0
cmm005	173	404	182.1 ± 2.7	0.594	11.2	178.2	74.0
020.br17.10	17	88	44.8 ± 0.8	0.011	0.2	44.3	44.3
020.br17.12	17	92	44.2 ± 0.1	0.002	0.2	44.1	44.1
020.ESC12	13	64	1389.8 ± 0.0	0.000	0.1	1389.8	1389.8
020.ESC25	26	134	1388.2 ± 11.7	0.004	0.4	1383.1	1383.1
020.ESC47	48	245	1204.4 ± 46.7	0.144	1.5	1062.6	1030.4
020.ESC63	64	350	50.5 ± 0.1	0.018	3.5	50.4	49.6
020.ESC78	79	414	14936.1 ± 47.3	0.227	4.0	14425.0	11540.0
020.ft53.1	53	282	6265.5 ± 49.4	0.038	1.9	6197.4	6024.8
020.ft53.2	53	275	6940.6 ± 26.3	0.075	1.8	6717.8	6420.8
020.ft53.3	53	282	8863.2 ± 109.9	0.074	1.6	8718.3	8209.6
020.ft53.4	53	276	11973.5 ± 71.5	0.013	1.6	11823.2	11823.2
020.ft70.1	70	346	32996.3 ± 113.3	0.047	3.4	32794.7	31450.4
020.ft70.2	70	351	34381.7 ± 314.2	0.067	3.2	33613.7	32080.8
020.ft70.3	70	347	36129.4 ± 415.3	0.058	2.9	35532.5	34028.0
020.ft70.4	70	353	45038.1 ± 167.2	0.049	2.7	44847.0	42824.0
020.kro124p.1	101	515	34047.8 ± 375.0	0.089	8.5	33509.7	31010.0
020.kro124p.2	101	525	35775.1 ± 571.9	0.109	8.0	34775.7	31873.0
020.kro124p.3	101	535	43362.4 ± 508.9	0.190	7.4	42510.8	35123.0
020.kro124p.4	101	527	65755.2 ± 758.5	0.112	6.6	64491.5	58417.0
020.p43.1	43	204	22613.8 ± 17.6	0.005	1.1	22574.9	22512.0
020.p43.2	43	199	22875.8 ± 7.1	0.004	1.0	22854.1	22784.0
020.p43.3	43	212	23190.9 ± 12.0	0.005	1.0	23174.7	23068.0
020.p43.4	43	205	66956.5 ± 42.6	0.002	0.9	66848.4	66848.4
020.prob42	41	208	209.9 ± 4.4	0.074	1.0	202.0	194.4
020.prob100	99	510	1359.7 ± 39.1	0.312	7.2	1291.0	936.0
020.rbg048a	49	255	287.2 ± 0.6	0.022	1.4	286.4	280.8
020.rbg050c	51	259	384.0 ± 1.5	0.027	1.5	380.7	373.6
020.rbg109a	110	573	862.3 ± 4.8	0.037	7.8	856.9	830.4
020.rbg150a	151	871	1448.4 ± 5.8	0.033	19.5	1440.1	1400.0
020.rbg174a	175	962	1681.5 ± 4.0	0.033	27.2	1678.1	1626.4
020.rbg253a	254	1389	2440.6 ± 8.0	0.033	86.4	2430.2	2360.0
020.rbg323a	324	1825	2617.2 ± 10.1	0.040	235.6	2601.4	2512.0
020.rbg341a	342	1821	2248.3 ± 12.6	0.086	246.5	2237.3	2054.4
020.rbg358a	359	1967	2200.9 ± 15.3	0.075	301.9	2178.6	2036.0

Table 5.1: Updated results for the HACS heuristic *(continued)*.

Instance	m	n	HACS			Best known	
			Mean ± SD	Gap	T (s)	UB	LB
020.rbg378a	379	1974	2421.5 ± 14.4	0.072	322.3	2392.7	2247.2
020.ry48p.1	48	256	13308.5 ± 72.9	0.050	1.5	13151.5	12644.0
020.ry48p.2	48	250	14003.5 ± 123.1	0.082	1.4	13804.6	12859.2
020.ry48p.3	48	254	16901.2 ± 184.8	0.077	1.3	16612.1	15592.0
020.ry48p.4	48	249	26275.0 ± 146.6	0.011	1.2	25980.0	25980.0
Averages			13754.0 ± 197.7	0.081	30.5		

The results show a significant improvement in execution time compared to the results in Paper I. For example, cmm005 took over 600 seconds on average in Paper I and is now 50 times faster. This makes the HACS algorithm better than the currently used PCGTSP heuristic in IPS with respect to both solution quality (by over 12% on average for cmm005) and execution time.

On average the mean solution value is at least within 10% of the optimal solution, and the average relative standard deviation is within 2%. For the instances cmm004 and cmm005 there is probably a lot of slack in the best known lower bound estimation and therefore the gap is quite large. For some of the SOP derived problem instances, such as 020.ESC78, 020.kro124p.3, and 020.prob100, the algorithm is shown to perform significantly worse than average. Further investigation of why these problem instances are harder to solve is needed.

6. Conclusions and Future Work

The heuristic algorithm presented in this thesis is, on average, able to produce good solutions for single robot stations within a reasonable time frame. The exact algorithm is able to solve some of the instances which are smaller and more dense with precedence constraints. The evaluation of the bounding methods shows that the assignment problem bound is more efficient than the bound obtained by solving a minimum spanning arborescence problem. The bound obtained from the assignment problem based on L-paths is shown to be of varying quality depending on the data but does not seem to consistently give better bounds with increasing values on L.

Because of a lack of exact methods and valid lower bounds for the test instances in the multiple agent case, assessment of the quality of solutions produced by the HACS algorithm is not possible. Comparison with the current solver in the IPS software shows an improvement around 2% on average. It can however be concluded that the calculations of the makespan and delays become more cumbersome as the number of precedence constraints increases. In particular, the local search procedure which evaluates many candidate solutions in the local search neighborhoods becomes significantly slower.

While the HACS algorithm for the single agent PCGTSP is proven to produce acceptable results on average, vertex selection improvement which is now only considered fairly scarcely in the algorithm process could be incorporated more. For example, some estimation of vertex selection when evaluating a feasible 3-exchange could be considered.

The branch-and-bound algorithm for the PCGTSP may be improved in many ways. The bounding methods that are used involve defining simpler problems where the precedence constraints are almost completely relaxed. This weakens the lower bounds considerably since the precedence constraints are often pivotal in defining the feasible region of the PCGTSP. To remedy this, one needs to take the precedence constraints into account in the bounding process. One of the aims of the L-paths were to reintroduce them and the vertex selection constraints into the bounding method but instances with vertices which are particularly cheap to visit made the L-distances very short, and therefore the lower bounds became quite weak. The L-distances could be strengthened by utilizing a modified branching strategy where cheap vertices are identified and branched on. Finally, dualization of constraints coupled with a subgradient method could further strengthen the lower bounds. Even though initial tests with dualization of the vertex degree constraints coupled with a simple subgradient method did not give very good results, more sophisticated methods may prove to be successful. Strengthening the bound at the root could be particularly beneficial since this gives a better estimation of the quality of the best feasible solution found by the algorithm.

In order to better assess the solutions produced by the HACS algorithm for the PCGmTSP, there is a need to develop exact methods for obtaining optimal solutions or lower bounds. While the algorithm seems to consistently produce better solutions when the local search procedure is allowed to more thoroughly explore its neighborhood, the time it requires to do so increases considerably for problem instances where many inter agent constraints arise. To adapt, one could opt for a more restricted local search procedure in cases with many inter agent constraints, or even forgo the local search procedure completely and instead only rely on the explorative nature of the ant path generation itself.

7. Summary of Publications

Paper I - An industrially validated CMM inspection process with sequence constraints

Authors: R. Salman, J.S. Carlson, F. Ekstedt, D. Spensieri, J. Torstensson, R. Söderberg.

This conference paper presents a heuristic approach for approximating the single agent PCGTSP based on the Hybridized Ant Colony System algorithm [23]. It also gives more detailed insight on how the PCGTSP arises in industrial applications, in particular in the process of computer generated coordinate-measuring machine programs.

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

Authors: R. Salman, F. Ekstedt, P. Damaschke.

This paper showcases the results of an exact branch-and-bound based approach to the single agent PCGTSP. Different bounding methods are evaluated and a novel branching technique which utilizes dynamic programming is presented. A pruning technique previously developed for the SOP is also generalized and applied to the PCGTSP. This manuscript has been submitted to Discrete Optimization.

Paper III - A Hybridized Ant Colony System Approach to the Precedence Constrained Generalized Multiple Traveling Salesman Problem

Authors: F. Ekstedt, R. Salman, D. Spensieri.

This paper extends the work presented in Paper I to the multiple agent case. A more general local search procedure is incorporated into the Hybridized Ant Colony System algorithm and more local search neighborhoods are explored. This manuscript has yet to be submitted for publication at the time of printing this thesis.

Bibliography

[1] A.I. Ali and J.L. Kennington, *The asymmetric M-travelling salesmen problem: A duality based branch-and-bound algorithm*, Discrete Applied Mathematics 13(2–3) (1986), pp. 259-276.

[2] 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.

[3] 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.

[4] E. Balas, *Machine Sequencing via Disjunctive Graphs: An Implicit Enumeration Algorithm*, Operations Research 17(6) (1969), pp. 941-957.

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

[6] E. Balas and N. Christofides, *A Restricted Lagrangean Approach to the Traveling Salesman Problem*, Mathematical Programming 21 (1981), pp. 19-46.

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

[8] E. Benavent and A. Martínez, *Multi-depot Multiple TSP: a polyhedral study and computational results*, Annals of Operations Reasearch 207(1) (2013), pp. 7-25.

[9] L. Bianco, A. Mingozzi, S. Ricciardelli, M. Spadoni, *Exact and Heuristic Procedures for the Traveling Salesman Problem with Precedence Constraints, Based on Dynamic Programming*, INFOR 32(1) (1994), pp. 19-32.

[10] J. Blazewicz, W. Domschkeb, E. Pesch, *The job shop scheduling problem: Conventional and new solution techniques*, European Journal of Operational Research 93(1) (1996), pp. 1-33.

[11] A. van Breedam, *Improvement Heuristics for the Vehicle Routing Problem based on Simulated Annealing*, European Journal of Operational Research 86 (1995), pp. 480-490.

[12] 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.

[13] 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.

[14] N. Christofides, A. Mingozzi, P. Toth, *State-Space Relaxation Procedures for the Computation of Bounds to Routing Problems*, Networks 11(2) (1981), pp. 145-164.

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

[16] 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.

[17] 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.

[18] 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.

[19] M. Fischetti, *Facets of the Asymmetric Traveling Salesman Polytope*, Mathematics of Operations Research 16(1) (1991), pp. 42-56.

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

[21] 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.

[22] M. Fischetti and P. Toth, *A Polyhedral Approach to the Asymmetric Traveling Salesman Problem*, Management Science 43(11) (1997), pp. 1520-1536.

[23] 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.

[24] B. Gavish and K. Srikanth, *An Optimal Solution Method for Large-Scale Multiple Traveling Salesmen Problems*, Operations Reasearch 34(5) (1986), pp. 698-717.

[25] A.H. Gharehgozli, G. Laporte, Y. Yu, R. de Koster, *Scheduling Twin Yard Cranes in a Container Block*, Transportation Science 49(3) (2017), pp. 685-705.

[26] J.A.S. Gromicho, J. Paixão, I. Bronco, *Exact Solution of Multiple Traveling Salesman Problems*, Combinatorial Optimization Vol. 82 (1992), pp. 291-292.

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

[28] 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.

[29] 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.

[30] M. Held and R. M. Karp, *The Traveling Salesman Problem and Minimum Spanning Trees*, Operations Research 18(6) (1970), pp. 1138-1162.

[31] M. Held and R. M. Karp, *The Traveling Salesman Problem and Minimum Spanning Trees: Part II*, Mathematical Programming 1 (1971), pp. 6-26.

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

[33] K. Helsgaun, *General k-opt submoves for the Lin–Kernighan TSP heuristic*, Mathematical Programming Computation 1(2–3) (2009), pp. 119–163.

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

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

[36] 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.

[37] S. Lin and B.W. Kernighan, *An Effective Heuristic Algorithm for the TravelingSalesman Problem*, Operations Research 21(2) (1973), pp. 498-516.

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

[39] 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.

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

[41] J.V. Potvin, G. Lapalme. J-M. Rousseau, *A Generalized K-Opt Exchange Procedure For The MTSP*, INFOR 27(4) (1989), pp. 474-481.

[42] 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.

[43] H.D. Sherali and P.J. Driscoll, *On Tightening The Relaxations Of Miller-Tucker-Zemlin Formulations For Asymmetric Traveling Salesman Problem*, Operations Research 50(4) (2002), pp. 656-669.

[44] 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.

[45] 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.

[46] 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.

[47] S. Somhom, A. Modares, T. Enkawa, *Competition-based neural network for the multiple travelling salesmen problem with minmax objective*, Computers & Operations Research 26 (1999), pp. 395—407.

[48] P. Toth and D. Vigo, *The Vehicle Routing Problem*, Society for Industrial and Applied Mathematics (2002).

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

Paper I

Available online at www.sciencedirect.com

SciVerse ScienceDirect

Procedia CIRP 00 (2016) 000–000

www.elsevier.com/locate/procedia

6th CIRP Conference on Assembly Technologies and Systems (CATS)

An industrially validated CMM inspection process with sequence constraints

Raad Salman[a,*], Johan S. Carlson[a], Fredrik Ekstedt[a], Domenico Spensieri[a], Johan Torstensson[a], Rikard Söderberg[b]

[a]*Fraunhofer-Chalmers Centre, Chalmers Science Park, SE-412 88 Göteborg, Sweden*
[b]*Wingquist Laboratory, Chalmers University of Technology, 412 96 Gothenburg, Sweden*

[*] Corresponding author. Raad Salman, Tel.: +46-739-837573; *E-mail address:* raad.salman@fcc.chalmers.se

Abstract

An efficient CMM inspection process implemented in industry gives significant productivity improvements. A key part of this improvement is the optimization of the inspection sequences. To ensure quality of the inspection the sequences are often constrained with respect to the order of the measurements. This gives rise to so called precedence constraints when modelling the inspection sequence as a variation of the travelling salesperson problem (TSP). Two heuristic solution approaches and a generic optimizing algorithm are considered. A generation based stochastic algorithm is found to reduce cycle time by as much as 12% in comparison to the currently used algorithm.

Keywords: CMM; inspection; automation; task sequences; precedence constraints; generalized travelling salesman problem; sequential ordering problem

1. Introduction

Many products such as car and truck bodies, engines, medical prosthesis, mobile phones, and lumbering equipment depend visually and functionally on its geometry. Since variation is inherent in all production processes, consistent efforts in styling, design, verification and production aiming at less geometrical variation in assembled products, is a key to shortening development time of new products, as well as for choosing an efficient and resource-economic production process. The activities aiming at controlling geometrical variation throughout the whole product realization process are called the geometry assurance process. Figure 1 shows a general model for product realization consisting of a concept phase, a verification phase and a production phase.

The geometry assurance process, as defined in [1], relies on inspection data in all phases. Product concepts are analyzed and optimized to withstand the effect of manufacturing variation and tested virtually against available production data often based on carry over type of inspection. In the verification and pre-production phase the product and the production system is physically tested and verified. Adjustments are made to both product and production system based on inspection data. In full production the focus is to control the process and to detect and correct errors by analyzing inspection data. These inspection data are often collected before, during and after important assembly steps. In this way, important assembly issues as part, fixture and joining errors can be detected and corrected in an efficient manner.

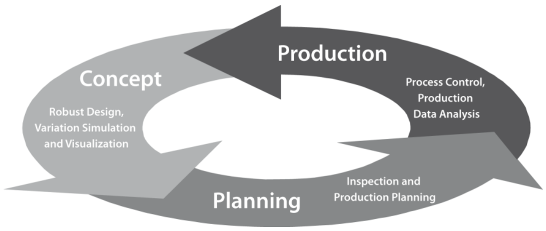

Production
Process Control, Production Data Analysis
Concept
Robust Design, Variation Simulation and Visualization
Planning
Inspection and Production Planning

Fig. 1. A general model for product realization and the main activities of the geometry assurance process.

Therefore, the inspection preparation and measuring is an important activity and this paper presents an industrial validated closed loop from inspection preparation to automatic efficient off-line programming of automated measurement equipment. Then the focus is on improving the sequence optimization part of it by solving precedence constrained generalized travelling salesperson problem.

2. An Efficient Process for Inspection Preparation and Programming

The efficient inspection process implemented to support programming of automated inspection devices is built up by five main steps; (i) define the inspection task by breaking down product and process requirements to geometrical inspection features, e.g. a hole or a slot, on part and subassembly level (Figure 2), (ii) create parameterized inspection rules that define how a feature should be measured, i.e. number of points, distribution, coordinate system, and probe cones, (iii) perform feature accessibility analysis to find a set of probe configurations of minimum size that can reach all inspection points with collision free CMM configurations (Figure 3), (iv) plan by math based algorithms for motion planning and combinatorial optimization the collision free motions and sequence of the measurement equipment to visit each feature, and (v) generate the control code, e.g. DMIS to instruct the equipment to perform the actual measurement.

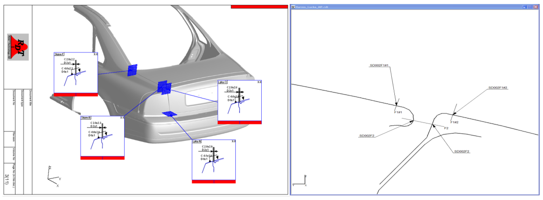


Fig. 2. An inspection task is defined by breaking down the product quality appearance requirement (right picture) on gap and flushes to boot and rear fender part inspection points (right picture).

This process has been industrially evaluated and used by e.g. Volvo Cars to program all automated inspection devices since 2011. The results show an improvement in inspection preparation time of 75% and productive increase in equipment utilization of 25%. The experience is also that the inspection preparation process becomes more structured and thereby reusable to a larger extent than previously.

2.1. *Parameterized Inspection Rules*

As mentioned, part of the process is to create parameterized inspection rules for the most commonly used inspection features in practice, i.e. surface point, edge point, circular hole, oval hole, rectangular hole, sphere, and cylinder [2,3]. The parameterization describes the inspection rule in terms of number of points, positions and probe configurations, and the allowed deviation from the ideal/default rule [4]. Today, it is common that the CMM embedded software contains the inspection rules and decides the motion patterns and sequence during feature inspection. However, the proposed approach with parameterized inspection features has four key advantages: (i) it makes the inspection preparation flexible, structured and repeatable, (ii) the same control code can be used with CMMs of different brands with more consistent results, (iii) the inspection sequence inside and between features can be optimized together to minimize cycle time, (iv) if the default inspection rule is not feasible due to collisions then the conflict can automatically be resolved by using the allowed deviation from the default rules. In Figure 4, as an example, the parameterized inspection rule for a circle is defined and illustrated.

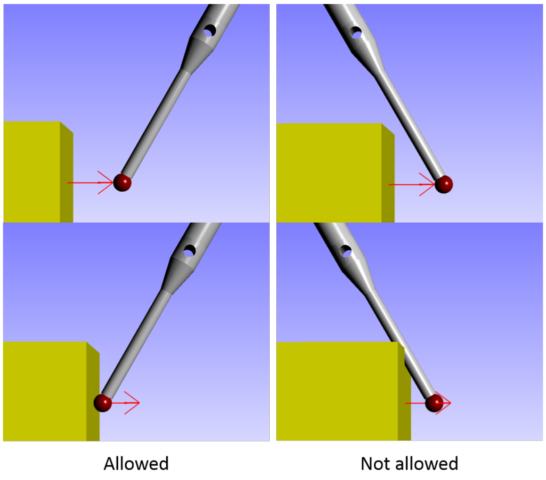


Fig. 3. Approachability illustrated; It should be possible to perform a linear motion along the inspection direction from a specified approach point and that the probe sphere/tip should make contact with the inspection point during that motion without any further collisions. The red arrow represents the normal of the inspection point.

Inspection Feature	Feature Definition	Inspection Rule
Circle	P: Position vector D: Diameter T: Thickness	t1: Measurement depth α1: Start angle (points on) plane α2: Start angle (points in) circle n1: #Points on plane n2: #Points in circle d1: Diameter on circle in plane

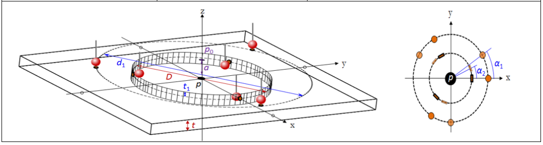


Fig. 4. A parameterized inspection rule of circle feature.

2.2. *Automatic Path Planning*

The next technology used is path planning where the collision free CMM motions are generated by automatically finding via points and probe reorientations between the inspection features [5,6,15]. Complete path planning algorithms, which always find a solution or determine that none exist, are of little industrial relevance since they are too slow. In fact, the complexity of the problem has proven to be PSPACE-hard for polyhedral object with polyhedral obstacles [7]. Therefore, sampling based techniques trading completeness for speed and simplicity is the choice. Common for these methods are the needs for efficient collision detection, nearest neighbor searching, graph searching and graph representation. The two most popular methods are; Probabilistic Roadmap Methods (PRM) [8] and Rapidly-Exploring Random Trees (RRT) [9]. These methods have been extended and tailored in several ways, for example in [10]. Inspired by these probabilistic methods FCC has developed a deterministic path planner that adaptively adjusts a grid in the configuration space.

2.3. Inspection Sequence Optimization

Data generated by the inspection rule analysis and path planning can then be used to optimize task sequences for robot stations, such as automated welding or measuring. Such optimization can reduce cycle time by as much as 25% and thereby greatly increase efficiency of production [11]. Task sequences can be discretized and modelled as a travelling salesperson problem (TSP) or some variation of it [12]. Introducing increasingly complex attributes to the problem such as different ways to complete each task, precedence constraints and/or several robot arms working on the same object requires the TSP model to be more advanced.

The precedence constraints are introduced by hierarchical relations between features since some features are required to be measured in relation to other features. Typically, to be able to measure some features there is a need for a local alignment. The alignment is a coordinate system calculated from group of measured/actual features. This type of local measurement creates hierarchical relations between features and thus imposes precedence constraints. However, this should not be confused with evaluating features in relation to each other. Sequence constraints are only introduced when features are physically measured in relations to other features. Spitz and Requicha [14] solved a constraint satisfaction problem to handle this. This paper will instead incorporate this directly in the TSP solution.

Therefore, this paper consider the case of optimizing the precedence constrained task sequence of a single arm CMM robot station where each task can be performed in several different ways. Since a CMM has five degrees of freedom, each point can be approached from many different angles and thus be evaluated in a multitude of ways. To model these characteristics, one can discretize a subset of the different ways in which a point can be measured and constrain the order of the points being evaluated. Given such a discretization and set of precedence constraints one can model the problem of minimizing the total cycle time as a precedence constrained generalized travelling salesperson problem (PCGTSP).

Since the PCGTSP is an extension of the GTSP it is also an NP-hard problem [17]. So as with many other NP-hard problems, using exact optimizing algorithms for solving larger problem instances are often impractical and heuristic algorithms are implemented instead [23]. The PCGTSP is similar to two other well-studied variations of the TSP, the sequential ordering problem (SOP) [21–26] and the generalized TSP (GTSP) [16–20], but the PCGTSP has not been extensively studied itself. Therefore, there is a need to develop and evaluate heuristic algorithms for the PCGTSP and their effectiveness on real industrial applications which is what this paper aims to do.

2.4. Results from Volvo Cars

At Volvo Cars a new vehicle program is inspected with typically 700 inspection programs containing up to 25 000 features. By implementing this efficient process for inspection the preparation and programming time have been estimated to be reduced by 75% and the equipment utilization has been improved by 25% more efficient programs. Some examples from the inspection process implementation at Volvo Cars can be seen in Figures 5-7.

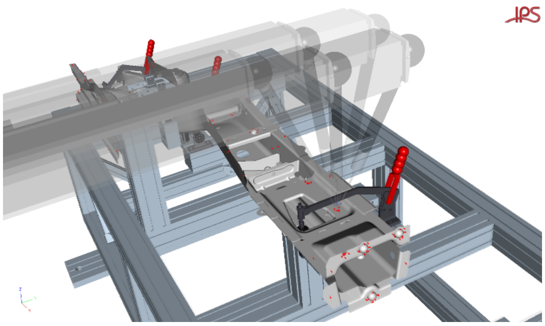


Fig. 5. Feature accessibility analysis resulting in five different collision free probe configuration inspection alternatives (courtesy of Volvo Cars).

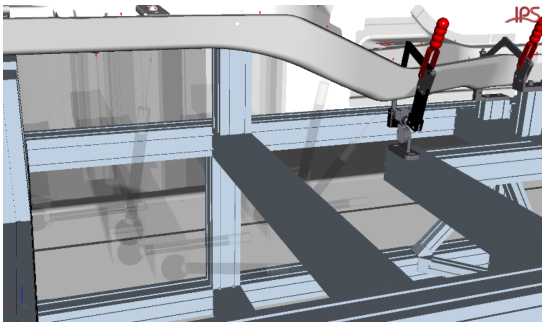


Fig. 6. An automatic generated collision free path between two features containing a non-trivial necessary probe change in the middle. Movement shown by transparent probe states (courtesy of Volvo Cars).

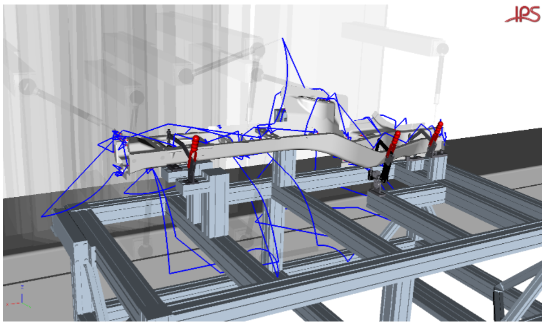


Fig. 7. An optimized collision free inspection sequence (blue trajectory) for 20 features containing 115 points, calculated by the system (courtesy of Volvo Cars).

The rest of the paper will proceed as follows. In Section

3 the PCGTSP is described. Section 4 describes the different solving methods which are evaluated in this paper and Section 5 presents the results when testing these methods on some real industrial cases. Finally, Section 6 presents some final conclusions and suggestions for future research.

3. Problem description

The PCGTSP is a variation of the TSP where the node set is partitioned into groups and then precedence constraints are enforced on a group level, i.e. such that the groups are required to precede each other (but not necessarily directly) in a solution. Because the PCGTSP solution represents a sequence of tasks (modelled as groups) where each task can be performed in different ways (modelled as nodes) it is natural to have the precedence constraints enforced on a group level, since the tasks are required to precede each other.

Let n be the number of nodes in a problem instance and let $V := \{1, \ldots, n\}$ denote the set of all nodes. Let $A := \{(i, j) : i, j \in V, i \neq j\}$ denote the set of all (directed) arcs between all nodes and let c_{ij}, $i, j \in V$, denote the cost associated with the arc from node i to node j. Let $M := \{1, \ldots, m\}$ denote the set of all group indices and let $V_1, \ldots, V_m$ be a partition of V where V_p, $p \in M$, is called a *group*. The partition of V must satisfy $V_p \neq \emptyset$, $V = \cup_{p \in M} V_p$ and $V_p \cap V_q = \emptyset$ when $p \neq q$. Let the precedence constraints be defined by sets which are denoted as $\mathrm{PG}_q := \{p \in M : \textit{group } p \textit{ must precede group } q \textit{ in the tour}\}$, $q \in M$. For these applications a start group, p_{start}, which consists of a single node is specified as the starting position of the robot as well. The PCGTSP is then to find a tour starting from p_{start} such that one node in every group is visited exactly once, the precedence constraints are satisfied and the sum of the cost associated with the traversed arcs is minimized.

When attempting to solve the PCGTSP one can view it as two subproblems: group sequence and node choice, i.e. the order in which the groups are visited and the choice of the node that is to be visited in each group. The group sequence subproblem requires a fixed selection of which node that is to be visited within each group while the node selection subproblem requires a fixed order of the groups to be solved. While there is a clear dependency between these subproblems, heuristic solving algorithms which separate or combine them to different degrees have, however, been shown to be efficient for the GTSP without precedence constraints [19,20].

4. Solution approaches

In this paper two different approaches for solving the PCGTSP are presented. The first approach is a deterministic algorithm which successively expands the set of groups as their precedence constraints are satisfied and uses a high performance heuristic algorithm designed for the GTSP as a lower level solver. The second approach is a stochastic algorithm based on an Ant Colony System (ACS) metaheuristic hybridized with a special purpose local search. This algorithm has been very successful for the SOP [23,24] and was shown to perform quite well for larger problem instances when generalized to the PCGTSP [13]. The generic optimizing software CPLEX is also considered as a solution approach and as a method for obtaining lower bounds.

4.1. CPLEX software

The CPLEX solver uses an advanced but generic method of branch-and-cut to optimize mixed integer linear programming (MILP) formulations of optimization problems. The MILP formulation of the PCGTSP first proposed by Salman in [13] is used to study the effectiveness of such a generic optimizing method in the industrial cases considered in this paper. CPLEX is used for completely solving the PCGTSP to optimality as well as solving the linear programming (LP) relaxed PCGTSP where the integrality constraints are relaxed and a lower bound on the minimal tour length is obtained.

4.2. Sequentially Expanding GTSP (SEG) solver

The general algorithm for the SEG solver is as follows:

Algorithm 1 Sequentially Expanding GTSP

```
1. Set $k = 1$ and initialize a path $P^1 = \{p_{\text{start}}\}$.
2. Set $U = \{p \in M$ : group $p$ is allowed to be visited
    given the path $P^k\}$.
3. Let the GTSP solver expand the path
   $P^k = \{P^k_1, P^k_2, \cdots, P^k_l\}$, $l \leq m$ using the groups in $U$.
4. If $P^k$ visits all groups in $M$ then add a final arc between
   $P^k_m$ and $P^k_1$ to the path $P^k$, reoptimize the node selection
   and exit.
5. For each $j = 1, \cdots, l$ check if any groups in $M$ are allowed
   to be visited given the path $P^{k_j} = \{P^k_1, \cdots, P^k_j\}$. As soon
   as one or several groups in $M$ are allowed to be visited for
   some $P^{k_j}$ then set $P^{k+1} = P^{k_j}$, set $k = k + 1$ and go to step
   2.
```

The SEG solver approach handles the precedence constraints implicitly and is also constructive in its nature, meaning that it is deterministic and does not iteratively improve the solution. The benefit of the SEG algorithm is that any GTSP solver can be used in conjunction with this general strategy and one can therefore utilize the many effective solving algorithms which have been developed for the GTSP. A potential drawback is the short-sightedness of the algorithm since it only considers the groups allowed to be visited in the graph given a current path constructed by the GTSP solver.

4.3. Hybridized Ant Colony System (HACS)

The idea for the ACS algorithm is to model a fixed number of ants, N, that iteratively generate feasible solutions to the PCGTSP by traversing arcs, $(i, j) \in A$, in a non-deterministic manner. In each iteration the generation of paths is guided by the depositing of "pheromones", which are denoted $\tau_{ij} \in [0, 1]$, along the arcs that have been traversed by the ant which has produced the shortest tour. The higher the value of τ_{ij}, the higher the probability that arc (i, j) is chosen during the process of generating paths. For each arc $(i, j) \in A$ a fixed parameter $\eta_{ij} \in [0, 1]$ is initialized as $\eta_{ij} = 1/c_{ij}$. This parameter is called the visibility parameter and provides a fixed measurement of how attractive the corresponding arc is for the ants.

However, to avoid getting stuck at locally optimal solutions and to promote diverse solutions the ACS algorithm incorporates a so-called evaporation rate parameter $\rho \in [0,1]$. Let $T^k = (T^k_1, \ldots, T^k_m)$ be the shortest tour in iteration k. At the end of each iteration k the pheromone levels are updated as $\tau_{ij} = (1-\rho)\tau_{ij} + \rho/C_{T^k}$ where C_{T^k} is the total cost of tour T^k. Furthermore, during the path generation process, if an ant chooses to traverse an arc (i,j), the pheromone level of that arc is updated as $\tau_{ij} = (1-\rho)\tau_{ij} + \rho\tau_0$ where τ_0 is the initial pheromone level parameter. The ACS algorithm also introduces a probability $d_0 \in [0,1]$ that the arc chosen by an ant during the path generation is the arc which is the most attractive. Otherwise, i.e. with probability $(1-d_0)$, an arc (i,j) is chosen with probability

$$p^a_{i,j} = \begin{cases} \frac{[\tau_{ij}]^\alpha [\eta_{ij}]^\beta}{\sum_{l \in V(T^a)} [\tau_{il}]^\alpha [\eta_{il}]^\beta} & \text{if } j \in V(T^a) \\ 0 & \text{otherwise} \end{cases} \qquad (1)$$

where α and β be parameters that control the relative importance of the pheromone level and the visibility parameter and $V(T^a)$ is the set of allowed nodes given a tour T^a of ant a.

4.3.1. Local search

After each tour generated by the ACS metaheuristic a local search procedure is executed. First, the node selection of the tour is fully optimized given a fixed order of the groups through a dynamic programming algorithm [13,16,19].

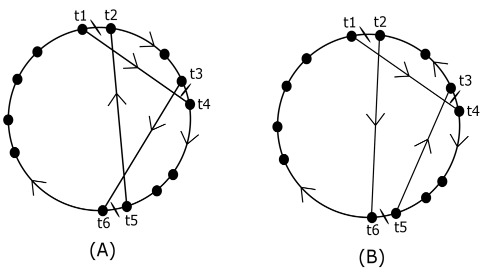


Fig. 8. (A) is an example of a path preserving 3-opt move. (B) is an example of a path inverting 3-opt move.

Then a highly efficient 3-opt local search [23] is performed. The k-opt local search heuristic removes k arcs from an existing tour and adds k arcs such that the tour becomes improved. This 3-opt local search was specifically developed to handle precedence constraints by excluding certain 3-opt moves from the search and was found to perform better than many other k-opt local search heuristics when generalized to the PCGTSP [13]. By excluding so-called path inverting 3-opt moves, i.e. moves that inverts the orientation of one or several segments of the tour (see Figure 8), the algorithm reduces the time spent on verifying that the precedence constraints are satisfied and verifying the improvement condition of a 3-exchange. Furthermore, the 3-opt local search employs a special labelling procedure which makes the verification of the precedence constraints even more efficient.

When a tour which can not be improved further by the 3-opt local search is found, the node selection is fully optimized again.

5. Computational experiments and results

Five problem instances derived from CMM inspection cases of various sizes are studied. Each problem instance is evaluated using the three solution approaches described in Section 4.

The CPLEX software was run with a 24 hour time limit and was run for the LP relaxed problem as well as the original MILP problem for each instance.

The HACS algorithm was run 10 times with 10 ants and 100 iterations per run. The parameters were set to $\rho = 0.1$, $\alpha = 1$, $\beta = 2$, $d_0 = 0.9$ and $\tau_0 = 1/(mC_u)$ where C_u is an upper bound on the minimal tour length. Also, the local search is only run for a generated tour if the cost is within 20% of the best one found so far. This heuristic rule as been found to be benificial in [24].

Let z be the sum of costs c_{ij} for the arcs (i,j) traversed in a solution. An optimal solution is then the shortest possible tour given the graph of a problem instance.

Table 1. Results from CPLEX. z^*_{LP} is the minimal solution for the LP relaxed problem and z^*_{MILP} is the minimal tour length for the original MILP problem. T_{LP} is the time for the LP relaxed problem and T_{MILP} is the time for the original MILP problem.

Instance	m	n	z^*_{LP}	T_{LP} (s)	z^*_{MILP}	T_{MILP} (s)
cmm001	13	15	48.85	0.03	49.12	0.03
cmm002	16	25	7.60	0.03	20.26	1.86
cmm003	18	36	11.43	0.19	20.04	0.41
cmm004	91	216	23.00	3936.43	-	>86400
cmm005	174	405	-	>86400	-	>86400

Table 1 shows the tour lengths when running the problem instances in the generic optimizing software CPLEX. For the two larger problems, cmm004 and cmm005, CPLEX was not able to find an optimal solution within the time limit of 24 hours and for cmm005 CPLEX was not able to solve the LP relaxation to optimality within the time limit either.

Table 2. Tour lengths and average running times for the heuristic algorithms. $z^{\mathrm{best}}_{\mathrm{HACS}}$ is the best (shortest) tour length out of 10 runs. T_{HACS} and T_{SEG} is the average time for completing a run.

Instance	m	n	z_{SEG}	T_{SEG} (s)	$z^{\mathrm{best}}_{\mathrm{HACS}}$	T_{HACS} (s)
cmm001	13	15	49.12	0.01	49.12	6.93
cmm002	16	25	20.48	0.02	20.73	9.56
cmm003	18	36	20.46	0.02	20.04	6.69
cmm004	91	216	48.31	2.80	46.07	286.91
cmm005	174	405	212.23	22.03	185.83	698.52

Table 2 shows the results from the heuristic algorithms. For the smaller instances, cmm001-003, the difference in solution quality is marginal. For cmm004 the HACS algorithm performs a bit better than the SEG solver and for cmm005 the solution produced by the HACS algorithm is significantly better. While the HACS algorithm is much slower than the SEG solver, Table 3 suggests that the number of iterations can probably be

Table 3. More detailed results for the HACS algorithm. $z^{\text{avg}}_{\text{HACS}}$ is the average solution found for an instance over 10 runs. T_{bavg} is the average running time and I_{bavg} is the average number of iterations elapsed before the best solution is found by the HACS algorithm in each run.

Instance	m	n	$z^{\text{best}}_{\text{HACS}}$	$z^{\text{avg}}_{\text{HACS}}$	T_{bavg} (s)	I_{bavg}
cmm001	13	15	49.12	49.12	0.02	1
cmm002	16	25	20.73	20.73	0.20	2
cmm003	18	36	20.04	20.11	1.36	13
cmm004	91	216	46.07	46.98	176.56	54
cmm005	174	405	185.83	187.61	237.63	33

lowered by almost 50% without any significant loss of solution quality for these problem instances.

6. Conclusions and future research

The productivity of the CMM inspection process and equipment is significantly improved by a structured inspection preparation process combined with automatic path planning. Inspection sequence optimization is an important part of the improvement. In this paper, the optimization part related to inspection sequence precedence constraints is further improved.

The presented HACS algorithm is able to reduce cycle time of the largest case by more than 10% on average in comparison to the now used SEG solver and while it is much slower, the number of iterations can probably be significantly tightened for the studied cases without losing much in terms of solution quality. The results from the CPLEX software shows the need for developing heuristic algorithms and special purpose optimizing algorithms for the PCGTSP.

Further development of the MILP model in conjunction with the optimizing algorithms might enable optimization of small to medium sized problem instances within reasonable computation times. For some industrial cases there arises a need for multiple CMMs evaluating features on the same object which corresponds to expanding the PCGTSP to a precedence constrained generalized multiple travelling salesperson problem (PCGmTSP).

Acknowledgements

This work was carried out within the Wingquist Laboratory VINN Excellence Centre, supported by the Swedish Governmental Agency for Innovation Systems (VINNOVA). It is also part of the Sustainable Production Initiative and the Production Area of Advance at Chalmers University of Technology.

References

[1] R. Söderberg, L. Lindkvist, J.S. Carlson, Virtual Geometry Assurance for Effective Product Realization, 1st Nordic Conference on Product Lifecycle Man- agement – NordPLM'06; 2006, Göteborg, Sweden.

[2] M.R. Henderson, D.C. Anderson, Computer Recognition and Extraction of Form Features: A CAD/CAM Link, Computers in Industry; 1984, vol. 5, pp. 329-339.

[3] H.C. Lee, W.C. Jhee, H. S. Park, Generative CAPP Through Projective Feature Recognition, Computers and Industerial Engineering; 2007, vol. 53, pp. 241-246.

[4] F. Zhao, X. Xu, S.Q. Xie, Computer Aided Inspection Planning: The state of the art, Computer Aided design; 2009, vol. 60(2), pp. 453-466.

[5] C.W. Ziemian, D.J. Medeiros, Automating probe selection and part setup planning for inspection on a coordinate measuring machine, International Journal of Computer Integrated Manufacturing; 1998, vol. 11(5), pp. 448-460.

[6] C.W. Ziemian, D.J. Medeiros, Automated feature accessibility algorithm for inspection on a coordinate measuring machine, International Journal of Production Research; 1997, vol. (35)10, pp. 2839-2856.

[7] J.F. Canny, The Complexity of Robot Motion Planning, MIT Press; 1988.

[8] R. Bohlin, L.E. Kavraki, Path Planning Using Lazy PRM, In: Proc. IEEE Int. Conf. On Robotics and Automation; 2000.

[9] S.M. LaValle, J.J. Kuffner, Randomized Kinodynamic Planning, In: Proc. IEEE Int. Conf. on Robotics and Automation; 1999.

[10] S. Karaman, E. Frazzoli, Sampling-based Algorithms for Optimal Motion Planning, International Journal of Robotics Research; 2011, vol. 30, iss. 7, pp. 846-894.

[11] J. Segeborn, D. Segerdahl, F. Ekstedt, J.S. Carlson, M. Andersson, R. Söderberg, "An Industrially Validated Method for Weld Load Balancing in Multi Station Sheet Metal Assembly Lines", ASME Journal of Manufacturing Science and Engineering; 2013.

[12] S. Björkenstam, D. Spensieri, J.S. Carlson, R. Bohlin, D. Gleeson, Efficient Sequencing of Industrial Robots through Optimal Control, Procedia CIRP 2014, vol 23, pp. 194-199.

[13] R. Salman (2015), Algorithms for the precedence constrained generalized travelling salesperson problem (Master's thesis), Retrieved from http://www.chalmers.se/en/departments/math/research/research-groups/optimization/Pages/master-thesis-projects.aspx.

[14] S.N. Spitz, A.A.G. Requicha, Hierarchical Constraint Satisfaction for Dimensional Inspection Planning, Proceedings of the 1999 IEEE International Symposium on Assembly and Task Planning, Porto, Portugal, July 1999.

[15] A.J. Spyridi, A.G. Requicha, Accessibility analysis for the automatic inspection of mechanical parts using coordinate measuring machines, International conference on Robotics and Automation; 1990, pp.1284-1289.

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

[17] I. Kara, H. Guden, O.N. Koc, New Formulations for the Generalized Traveling Salesman Problem, In Proceedings of the 6th International Conference on Applied Mathematics, Simulation, Modelling, ASM'12, pp. 60–65, Stevens Point, Wisconsin, USA, 2012. World Scientific and Engineering Academy and Society (WSEAS).

[18] D. Karapetyan, G. Gutin, Lin-Kernighan Heuristic Adaptation for the Generalized Trav- eling Salesman Problem, European Journal of Operational Research; 2011, vol 208, pp. 221–232.

[19] D. Karapetyan, G. Gutin, Local Search Algorithms for the Generalized Traveling Salesman Problem. European Journal of Operational Research; 2012, vol 219, pp. 234–251.

[20] G. Gutin, D. Karapetyan, N. Krasnogor, Memetic Algorithm for the Generalized Asymmetric Traveling Salesman Problem, In Nature Inspired Cooperative Strategies for Optimization (NICSO 2007), vol 129 of Studies in Computational Intelligence, pp. 199– 210, Springer Berlin Heidelberg; 2008.

[21] 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; 2005, vol 33, pp. 62–70.

[22] N. Ascheuer, M. Jünger, G. Reinelt, A Branch & Cut Algorithm for the Asymmetric Traveling Salesman Problem with Precedence Constraints, Computational Optimization and Applications; 2000, vol 17, pp. 61–84.

[23] L.M. Gambardella, M. Dorigo, An Ant Colony System Hybridized with a New Local Search for the Sequential Ordering Problem, INFORMS Journal on Computing; 2000, vol 12, pp. 237–255.

[24] L.M. Gambardella, R. Montemanni, D. Weyland, An Enhanced Ant Colony System for the Sequential Ordering Problem, In Operations Research Proceedings 2011, Operations Research Proceedings, pp. 355–360, Springer Berlin Heidelberg; 2012.

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

[26] J. Sung, B. Jeong. An Adaptive Evolutionary Algorithm for Traveling Salesman Problem with Precedence Constraints. The Scientific World Journal, 2014, 2014.

Paper II

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 $\sigma = (V_1, \dots, V_r)$. 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) $\hat{\sigma} = (j, v_{L+1}, v_{L+2}, \dots, v_m, i)$ 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($\mathcal{P}$)
 2:     LB$_{\text{best}}$ = 0
 3:     UB$_{\text{best}}$ = $\infty$
 4:     bestLeaf = NULL
 5:     Let Q be an empty stack
 6:     if $m < 20$ then
 7:         DynamicProgramming($\mathcal{P}$)
 8:     else
 9:         Let $\mathcal{N}_0$ be the root node
10:         Bound($\mathcal{N}_0$)                                   ▷ Bound the root node
11:         LB$_{\text{best}}$ = $\mathcal{N}_0$.LB
12:         Q.push($\mathcal{N}_0$)
13:         while Q is not empty do
14:             currentNode = Q.pop()
15:             if currentNode.IsLeaf & currentNode.UB < UB$_{\text{best}}$ then
16:                 bestLeaf = currentNode
17:                 UB$_{\text{best}}$ = currentNode.UB
18:             else if currentNode.LB < UB$_{\text{best}}$ 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

```
1: function BOUND($\mathcal{N}$)
2:     Let historySuffix[] be a global hashmap        ▷ Suffix part of lower bound for every $(S, r)$ pair
3:     $\sigma = \mathcal{N}$.GroupSequence
4:     Let $S(\sigma)$ be the set of groups in $\sigma$
5:     Let $p$ be the last group in $\sigma$
6:     if historySuffix$[(S(\sigma), p)] < \infty$ then
7:         lowerBound = historySuffix$[(S(\sigma), p)]$
8:     else
9:         if useNoonBean then
10:            if useArborescence then
11:                lowerBound = ComputeMinArborescence(NB($\mathcal{N}$.SubProblem))
12:            else
13:                lowerBound = ComputeMinAssignment(NB($\mathcal{N}$.SubProblem))
14:            end if
15:        else
16:            if useArborescence then
17:                lowerBound = ComputeMinArborescence(NC$_1$($\mathcal{N}$.SubProblem))
18:            else
19:                lowerBound = ComputeMinAssignment(NC$_L$($\mathcal{N}$.SubProblem))
20:            end if
21:        end if
22:        historySuffix$[(S(\sigma), p)]$ = lowerBound
23:    end if
24:    $\mathcal{N}$.LB = $\min(P_{ij}^{(\sigma)})$ + lowerBound
25: end function
```

Algorithm 3.3 Branching subroutine

```
1: function BRANCH(Q, $\mathcal{N}$)
2:     Let historyPathCosts[] be a global hashmap        ▷ Path costs for every $(S, r)$ pair
3:     Let N be an empty stack
4:     currentGroupSequence = $\mathcal{N}$.GroupSequence
5:     allowedGroups = $\mathcal{N}$.AllowedGroups    ▷ Contains all groups which are allowed to be sequenced after the
   last group in currentGroupSequence
6:     while allowedGroups is not empty do
7:         $p$ = FindSmallestSuccessorSet(allowedGroups)
8:         $\sigma$ = [currentGroupSequence, $p$]
9:         Let $\mathcal{N}(\sigma)$ be a new tree node
10:        Let $S(\sigma)$ be the set of groups in $\sigma$
11:        if $|S(\sigma)| = m$ then
12:            $\mathcal{N}(\sigma)$.IsLeaf = true
13:            $\mathcal{N}(\sigma)$.UB = ComputeUpperBound($\sigma$)    ▷ Applies three-opt heuristic to $\sigma$ and computes the
   cost of the resulting tour
14:        else if historyPathCosts$[(S(\sigma), p)]_{ij} < c(P_{ij}^{(\sigma)})$ ,$\forall (i, j) \in V_1 \times V_p$ then
15:            delete $\mathcal{N}(\sigma)$        ▷ Prune node
16:        else
17:            $\mathcal{N}(\sigma)$.SubProblem = $\mathcal{P}_2(\sigma)$
18:            Bound($\mathcal{N}(\sigma)$)
19:            for every $(i, j) \in V_1 \times V_p$ do
20:                if historyPathCosts$[(S(\sigma), p)]_{ij} > c(P_{ij}^{(\sigma)})$ then
21:                    historyPathCosts$[(S(\sigma), p)]_{ij} = c(P_{ij}^{(\sigma)})$
22:                end if
23:            end for
24:            Q.push($\mathcal{N}(\sigma)$)
25:        end if
26:        delete $p$ in allowedGroups
27:    end while
28: 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 $NC_1(\mathcal{P})$. Dynamic programming presolve step is enabled.

Instance	m	n	$\|\Pi\|$	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 $O(n^2 + m^2 n^2)$ 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.

Paper III

A Hybridized Ant Colony System Approach to the Precedence Constrained Generalized Multiple Traveling Salesman Problem

Fredrik Ekstedt[1*] fredrik.ekstedt@fcc.chalmers.se
Raad Salman[1] salman@fcc.chalmers.se
Domenico Spensieri[1] domenico.spensieri@fcc.chalmers.se

[*]Corresponding author.
[1]Fraunhofer-Chalmers Centre, Chalmers Science Park, 412 88 Gothenburg, Sweden

October 20, 2017

Abstract

Given an edge-weighted graph where the set of vertices has been partitioned into a number of pairwise disjoint and non-empty sets (groups), the Precedence Constrained Generalized Multiple Traveling Salesman Problem (PCGmTSP), as considered in this paper, requires a predetermined number of cycles (or tours) such that exactly one vertex in each group is visited exactly once, and the length of the longest tour is minimized. Additionally, the tours must be such that the order in which the groups are visited must respect some given precedence relations. These relations may cause a group which is required to succeed another group which is visited in another tour to be delayed. This paper presents an Ant Colony Optimization approach to solving the PCGmTSP. A local search procedure based on known heuristics for the Sequential Ordering Problem, Vehicle Routing Problem, and the machine scheduling problem is incorporated. A novel way of interpreting improvement in the local search heuristics is introduced and estimations of the delays are employed within the heuristics. The results show that the local search procedure is quite good at improving the solutions but that the computation time of the algorithm increases drastically for problems with many precedence constraints.

Keywords: generalized multiple traveling salesman problem; precedence constraints; sequential ordering problem; vehicle routing problem; machine scheduling; ant colony optimization

1 Introduction

The Precedence Constrained Generalized Multiple Traveling Salesman Problem (PCGmTSP) combines several extensions of the classical Traveling Salesman Problem (TSP). The extension to a Generalized TSP (GTSP) means that the vertices are partitioned into disjoint subsets (groups) and that only one vertex in each subset needs to be visited. The Precedence Constrained (PC) extension introduces constraints on the form that a certain vertex (or group in the generalized case) must be visited before another vertex (or group) in the sequence. Finally, the multiple agent extension means that there are several salesmen (hereby known as *agents*) which share the task of visiting all vertices (or groups) exactly once and returning to their respective start vertex. There are two ways of interpreting precedence constraints in a multiple agent setting. The simplest is to only consider precedence constraints between groups that are assigned to the same agent, intra agent constraints. This works in the same way as in the single agent case. An extension to include constraints between groups assigned to different agents - inter agent constraints - requires a notion of time to be involved in the model. We will use the extended interpretation and assume that the edge and vertex costs represent times for traversing edges and visiting vertices. The precedence constraints may then be expressed using the corresponding accumulated times in the individual agents' tours.

Applications of the PCGmTSP include robot stations where several robots work in parallel to complete a number of tasks, e.g. welding or measuring. Typically, the cycle time - the time for all robots to complete their tasks - is to be minimized. Since the robots are located at different positions, their processing times for tasks and movements between tasks may differ between robots. Further, many tasks may be performed in several ways, hence the generalized version. Finally, in some applications there are various sequence constraints that put limitations on the order in which tasks are performed. Precedence constraints require that a certain task must be postponed until another task is completed. The reason for this may differ; for welding, it could be that some stabilizing weldings have to be done before others may be considered, and for measuring machines there is often a need to measure some points of the geometry first in order to establish global and/or local coordinate systems.

1.1 Previous work

The PCGmTSP combines elements from many different extensions of the classical TSP. Each of them - generalization, multiple agents, precedence constraints - has been studied extensively. But to the best of the authors' knowledge, no one has yet combined them all as presented in this paper.

The extension to several agents (salesmen, vehicles or machines) has been considered in the scientific literature from several viewpoints. The Vehicle Routing Problem (VRP) [1] is a classical problem - or class of problems rather - modeling a fixed or variable number of vehicles which share the task of visiting a number of customers and returning to a predetermined depot. Usually the sum of the traveling costs is minimized, subject to capacity constraints (CVRP). But there are also versions with the so called minmax objective where the cost of the longest tour is minimized instead [3]. The VRP may be defined for a single depot, from where the whole vehicle fleet emanates and returns to, or for a variable or fixed number of depots which coincides with our formulation. For the TSP, the multiple salesmen extension is denoted mTSP and is basically a VRP without capacity constraints. Here too, minimizing the sum of traveling costs is normally considered, but minmax versions have been studied as well [4–8]. Another viewpoint comes from the scheduling literature with parallel machine scheduling with sequence dependent setup times [11, 12].

The addition of precedence constraints to the TSP leads to the PCTSP or the equivalent Sequential Ordering Problem (SOP) [13], both of which have been extensively studied. Exact algorithms have been considered in [14–16] where problem instances with as many as 700 vertices have been solved to optimality. Heuristic approaches have been considered in [17–19] with the path preserving 3-exchange proposed in [17] being particularly successful.

The generalization extension of the TSP [20] has also been widely studied over the years. Exact branch-and-bound based algorithms have been studied as far back as 1983 in [21] and then continued in [22–24]. Fischetti et al. consider a branch-and-cut algorithm for the symmetric case in [25]. Different metaheuristic and neighbourhood search approaches have been developed in [26–30].

When it comes to combinations of the above TSP extensions, the literature is a bit more sparse. The generalized VRP (GVRP) is fairly well studied, including the multi-depot version in [37, 38]. For the GmTSP, some work has been done in [39] and [40], where the latter deals with the minmax version. The PCGTSP has been considered in [34–36]. In [41], a scheduling problem with sequence dependent set-up times, precedence constraints, and unrelated parallel machines are considered. Eligibility constraints are also imposed on the machines - all tasks cannot be performed by all machines - but there is no Generalized structure. Basically the same problem, but without the eligibility constraints, PCmTSP, is considered in [42] where different MILP formulations are presented and compared.

The work closest to ours is found in [43] in the context of container transportation. It is basically a PCGmTSP with two agents, a minmax objective and some additional spatial constraints. The assignment of groups (requests) to agents (cranes) is predefined however, so the load balancing aspect is not considered. Solutions to the problem were obtained through an adaptive large neighbourhood search based on several removal and insertion heuristics.

1.2 Contribution and outline

The main contribution of this paper is to address the PCGmTSP in its most general form. Some known heuristics - Path Preserving 3-opt, String Move and Delay Removal - are adapted to the problem. These adaptations are aimed at node selection (generalization) and predicting delays due the combination of precedence constraints and multiple agents. Simultaneously, a new way of looking at improvements in local search is introduced, which may be useful particularily for minmax problems. Further, an ant colony algorithm is developed and combined with the adapted local heuristics. Finally, a class of testbed problem is constructed by expanding existing SOP test problems.

In Chapter 2, a formal definition of the PCGmTSP is given, with extra focus on precedence constraints in general and between agents in particular. The latter aspect is further considered in Chapter 3 where the disjunctive graph is introduced as a necessary tool to compute the minmax objective function. In Chapter 4 the Hybrid Ant Colony System is introduced. In Chapter 5, a slightly modified Local Search is presented with some novel features. In Chapter 6, the new test problems are described, and the results from extensive computational experiments are presented.

2 Problem description

We consider a directed graph (V, E), where $V = 1, \dots, N$ denotes the set of vertices, and $E \subseteq V \times V$ denotes the set of directed edges. The vertices are divided into disjoint non-empty subsets $V_1, \dots, V_M$ referred to as *groups*. For convenience, we may refer to a group by its integer index instead of the full set notation. Furthermore, we denote the index of a group which includes the vertex v as $G(v)$ so that $v \in V_{G(v)}$ always holds. Each vertex v is also associated with a unique agent $A(v) \in \{1, ..., K\}$ which is able to process it. We may have for some problem instances that all agents are able to process any of the tasks, i.e. visit any group, and for others that each agent can only visit a subset of the groups but there exists a feasible solution where every group is visited exactly once. Either way, we assume, without loss of generality, that the agents may share access to the same group but never to the same vertex. Finally, for each agent a there is a designated starting group $G_0(a)$ which only contains vertices associated with a, that is, $G(v) = G_0(a) \implies A(v) = a$.

For each edge $(i, j) \in E$ there is a corresponding edge processing time $T^{\text{edge}}_{i,j}$, and for each vertex $v \in V$ a processing time T^{vertex}_v. Due to the precedence constraints, we may not simply add the vertex times to all incoming (or outgoing) edge cost in the ordinary xTSP fashion. We need to separate the start and end time of each vertex in a solution. Yet, it is sometimes convenient to use the following notation

$$\widetilde{T}_{i,j} = T^{\text{edge}}_{i,j} + T^{\text{vertex}}_j.$$

Just as for the single agent PCGTSP [36], precedence constraints are expressed on a group level, and may thus be seen as a separate acyclic directed graph $(\{1, ..., M\}, \Pi)$, where $(g_1, g_2) \in \Pi$ means that g_1 must precede g_2. With multiple agents however, the interpretation of $(g_1, g_2) \in \Pi$ becomes somewhat more complicated and requires some further definitions. We begin by defining what constitutes a solution for the mGTSP without precedence constraints.

Definition 1. *By a* solution *s to an mGTSP instance we mean K vertex sequences $s_a = \{v^a_l\}_{l=0,\dots,L(a)-1}$, $a = 1, \dots, K$, called agent tours, such that*

- $G(s^0_a) = G_0(a)$,
- $A(v^l_a) = a$,
- *for each group g there is exactly one agent $a = A(s, g)$ and one tour index l such that $G(v^l_a) = g$.*

Note that slightly ambiguous use of the function A, but it will be clear from the context which version is meant and this will not cause any confusion.

Definition 2. *For a solution s, an agent a, and a group g we define agent tasks start and end times, group agent start and end times, agent tour time, and cycle time as*

$$\begin{aligned}
T_{\text{end}}(s, a, l) &= \sum_{i=0}^{l-1} \widetilde{T}_{s^i_a, s^{i+1}_a}, l = 1, \dots, L(a) - 1 \\
T_{\text{sta}}(s, a, l) &= T_{\text{end}}(s, a, l) - T^{\text{vertex}}_{s^l_a}, l = 1, \dots, L(a) - 1 \\
T_{\text{sta}}(s, g) &= T_{\text{sta}}(s, a, l) \text{ for } a \text{ and } l \text{ such that } G(s^l_a) = g \\
T_{\text{end}}(s, g) &= T_{\text{end}}(s, a, l) \text{ for } a \text{ and } l \text{ such that } G(s^l_a) = g \\
T_{\text{tot}}(s, a) &= T_{\text{end}}(s, a, L(a)) \text{ with the auxiliary definition } s^{L(a)}_a := s^0_a \\
T_{\text{C}}(s) &= \max_a \{T_{\text{tot}}(s, a)\}
\end{aligned}$$

Even though other objective functions such as the sum of the agent tour times may be equally interesting, we will focus entirely on the cycle time of the solution (also known as the makespan). That is, the objective is to minimize the longest agent tour time.

Using the above definitions, we may state a preliminary definition of precedence constraint fulfillment. The precedence constraint $(g_1, g_2) \in \Pi$ is fulfilled in a solution s iff $T_{\text{end}}(s, g_1) \leq T_{\text{sta}}(s, g_2)$. For groups within the same agent tour (intra agent constraints) this is a natural interpretation. However, for constraints affecting groups in different agent tours (inter agent constraints), one quickly realizes that this definition is unnecessarily restrictive. If $T_{\text{end}}(s, g_1) > T_{\text{sta}}(s, g_2)$ were to happen for some inter agent constraint $(g_1, g_2) \in \Pi$ one could simply add a waiting time to the processing time of g_2 such that $T_{\text{end}}(s, g_1) = T_{\text{sta}}(s, g_2)$. This constraint relaxation would thus add potential delays to the tours, but could also open up for better solutions in problem instances which are dense with precedence constraints. To avoid complicating the notation, we will assume from now on that the quantities in Definition 2 include delays.

The PCGmTSP may then finally be completely defined as an mGTSP where the objective is to minimize the cycle time T_{C} of the solution, and with soft precedence constraints as defined above.

3 Disjunctive graph representation

The relaxation of inter agent constraints to "soft constraints" with incurred delays complicates the computations of the individual agents' tour times and the cycle time. To better facilitate these computations we introduce a *disjunctive graph* representation of a PCGmTSP solution.

Definition 3. *Given a PCGmTSP instance and a feasible solution s, the corresponding disjunctive graph is a directed acyclic graph consisting of*

- *All vertices in s*
- *One vertex s_a^{end} for each agent serving as an "end vertex"*
- *All edges (s_a^l, s_a^{l+1}) in s - except for edges $(s_a^{L(a)-1}, s_0^a)$ - with edge costs $\widetilde{T}_{s_a^l, s_a^{l+1}}$*
- *Edges $(s_a^{L(a)-1}, s_a^{\text{end}})$ with edge costs $\widetilde{T}_{s_a^{L(a)-1}, s_a^0}$*
- *Edges of the form (v_1, v_2) where $v_1, v_2 \in s$, $(G(v_1), G(v_2)) \in \Pi$, and $A(v_1) \neq A(v_2)$ - named disjunctive edges - with zero processing times.*

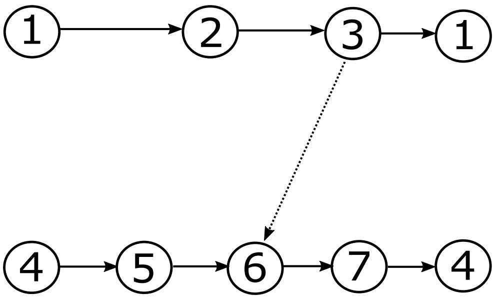


Figure 1: A disjunctive graph representation of a feasible PCGmTSP solution. Disjunctive edges are dotted.

The disjunctive graph representation facilitates the computations of the quantities $T_{\text{end}}(s, a, l)$ (and thereby $T_{\text{tot}}(s, a)$ and T_{C}) by the following known result [10].

Proposition 1. *The finishing time $T_{\text{end}}(s, a, l)$ is given by the longest path in the disjunctive graph ending at s_a^l.*

For a solution to be feasible, its disjunctive graph must be acyclic. This can be understood with the same logic which dictates that the precedence graph must be acyclic. Any solution to the PCGmTSP imposes a set of precedence constraints on the vertices involved which is what the conjunctive edges, all intra agent edges, actually represent. If the graph is cyclic after the disjunctive edges are added then the sequencing of the solution is incompatible with the precedence constraints which are imposed by the problem instance. Figure 2 illustrates the cyclic disjunctive graph of an infeasible solution. This sequence is impossible to perform since 1 must precede 2 which must precede 3 which must precede 6 which must precede 7 which must precede 2 etc. A solution which is infeasible because of such a phenomenon will be called *cyclic*.

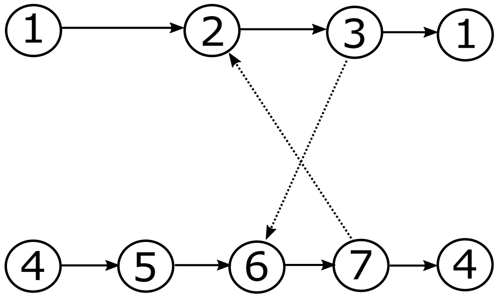


Figure 2: A disjunctive graph representation of an infeasible PCGmTSP solution. Disjunctive edges are dotted.

The computation of all finishing times $T_{\text{end}}(s, a, l)$ will be referred to as the Disjunctive Graph Longest Path algorithm or simply the DGLP. This includes checks for any cycles in the solution.

4 Hybridized Ant Colony System

The idea for the Hybridized Ant Colony System (HACS) algorithm is to model R generations of P ants, that iteratively generate feasible solutions by traversing edges, $(i, j) \in E$, in a non-deterministic manner. In each iteration during the solution generation, an ant is guided by the depositing of *pheromones*, denoted $\tau_{ij} \in [0, 1]$. The higher the value of τ_{ij}, the higher the probability that edge (i, j) is chosen to be traversed by the ant during that iteration. Also guiding the solution generation are fixed *visibility parameters*, $\eta_{ij} = 1/\widetilde{T}_{ij}$, which provide a fixed measurement of how attractive the corresponding edge is.

The pheromone levels contribute to the exploitation of good solutions. However, to avoid getting stuck at solutions which are locally optimal, the HACS algorithm incorporates a so-called evaporation rate parameter $\rho \in [0, 1]$ which controls the rate at which the pheromones along the edges evaporate. After each generation, i.e. when the P ants have generated one solution each, the pheromone levels are updated according to the global rule: $\tau_{ij} = (1 - \rho)\tau_{ij} + \rho/T_{\text{C}}(\bar{s})$ for every $(i, j) \in \bar{s}$, where $\bar{s}$ is the best solution found so far. Furthermore, during the solution generation process, if an ant chooses to traverse an edge (i, j), the pheromone level of that edge is updated according to the local rule: $\tau_{ij} = (1-\rho)\tau_{ij}+\rho\tau_0$ where τ_0 is the initial pheromone level parameter. The HACS algorithm also introduces a probability $d_0 \in [0, 1]$ that the edge chosen by an ant is the edge which is the most attractive, and chooses edges in proportion to η_{ij} and τ_{ij} with probability $(1 - d_0)$.

Let $\alpha, \beta \geq 1$ be parameters that control the relative importance of the pheromone level and the visibility parameter, respectively, when choosing an edge. The attractiveness of an edge $(i, j) \in E$ is then computed as:

$$\psi_{ij} = (\tau_{ij})^{\alpha}(\eta_{ij})^{\beta}. \tag{1}$$

An outline for the HACS framework and the procedure for generating a solution is given below.

Algorithm 1 HACS framework

```
1. Set $r := 1$ and set $T_{\mathrm{C}}(\bar{s}) = \infty$.
2. Set $p := 1$
3. Generate a solution $s_p^r$ according to solution generation algorithm and apply local search heuristics.
   If $T_{\mathrm{C}}(s_p^r) < T_{\mathrm{C}}(\bar{s})$ then set $\bar{s} = s_p^r$.
4. If $p < P$ then set $p := p + 1$ and go to step 3.
5. Set $r := r + 1$. Take the best solution found so far, $\bar{s}$, and update the pheromone levels as
   $\tau_{ij} = (1 - \rho)\tau_{ij} + \rho/T_{\mathrm{C}}(\bar{s})$ for every $(i, j) \in \bar{s}$.
6. If $r < R$ then go to step 2. Otherwise return overall best solution that was found and stop.
```

Algorithm 2 Solution generation for the HACS algorithm

```
1. Initialize the solution $s$ by setting the first vertex in each one of the agent tours to the predetermined
   start vertices and set $k := 2$.
2. Compute the set of vertices allowed to be sequenced next in the solution, $V(s)$, by taking into
   account the precedence constraints and the groups already visited in $s$.
3. Let $d \in [0, 1]$ be a uniformly distributed random number. If $d > d_0$ choose to traverse the edge
   $(i, j)$ with probability
```

$$f_{ij} = \begin{cases} \dfrac{\psi_{ij}}{\sum\limits_{l \in V(s)} \psi_{il}}, & \text{if } j \in V(s) \\ 0, & \text{otherwise.} \end{cases} \tag{2}$$

```
   If $d \leq d_0$ then let the edge $(i, j^*)$ be traversed where $j^* \in V(s)$ is such that $\psi_{ij^*} \geq \psi_{ij}$, i.e. the
   edge which is the most attractive is chosen.
4. If edge $(i, j)$ is traversed then set $s_{A(j)}^{L(A(j))-1} := j$ and update the pheromone levels according to
   $\tau_{ij} := (1 - \rho)\tau_{ij} + \rho\tau_0$.
5. If $k < M$ set $k := k + 1$ and go to step 2.
```

When computing the set of vertices allowed to be sequenced next in a partial solution s, $V(s)$, the inter agent constraints have to be fulfilled "sequentially" in order to avoid cyclic solutions. Which is to say, if $(G(v_1), G(v_2)) \in \Pi$ then $G(v_1)$ must be visited in s before $G(v_2)$ is allowed to be sequenced. This might seem as a severe rule to enforce, especially if all agents are able to access all groups. However, a relaxed rule where a group q is allowed to be sequenced on an agent if there exists other agents which can feasibly reach and sequence all groups $p : (p, q) \in \Pi$ such that no cycles are created, quickly becomes very complicated to verify. We will instead rely on the local search heuristics to improve the solutions in this slightly more limited space of solutions.

5 Local search procedure

In this section we will present a local search procedure for improving a given solution. It is based on three different heuristics:

- Path-preserving 3-opt (or simply 3-opt)
- String Move
- Delay Removal

The two first are adaptations of already well known local search heuristics for the SOP/PCTSP and/or mTSP/VRP, and the third is inspired by a known job shop scheduling heuristic. The adaptations are aimed at handling the two added complications; the group structure and the precedence constraints, with an emphasis on the latter.

5.1 Algorithm structure

The main algorithm for improving a given solution is given below. It is a classic descent search, but with a small twist. For reasons to be explained in Section 5.7, we will extend the notion of solution improvement by introducing an improvement measure I. We let $N(s)$ denote the set of solutions reachable by 3-opt, String Move and Delay Removal from solution s.

Algorithm 3 Local search - modified version

```
1. Find an initial solution $s_0$, set $s_* := s_0$ and $k := 0$
2. Search for a feasible $s_{k+1} \in N(s_k)$ such that $I(s_k, s_{k+1}) > 0$
3. If (2) fails or $k + 1 = k_{\max}$, terminate.
4. If $T_C(s_{k+1}) < T_C(s_*)$ set $s_* := s_{k+1}$
5. Set $k := k + 1$ and return to (2).
```

With a proper choice of improvement measure I, this modified local search will terminate after a finite number of steps. We have anyway imposed a maximum number of iterations, $k_{\max}$, since this algorithm will be run many times during HACS.

The neighbourhood search in step 2 is complicated by the fact that it is not always possible to easily predict the effect of local changes due to the inter agent precedence constraints, something that will be further explored below. Further, it is too expensive to invoke the DGLP for each solution in the neigbourhood $N(s_k)$ of the current solution. The following strategy will therefore be used for each heuristic and its corresponding neighbourhood separately.

Algorithm 4 Neigbourhood search

```
1. For each $s \in N(s_k)$, compute estimates of the agent tour times, and use these to estimate $I(s_k, s)$
2. Sort all $s \in N(s_k)$ according to the estimated improvements from (1)
3. Apply the DGLP to the most promising candidates of the sorted list from step 2 until an improve-
   ment has been found or a maximum number of candidates has been tried.
```

We would like to point out that the neigbourhoods $N(s_k)$ may contain infeasible cyclic solutions, since these are impossible to detect on the fly without invoking the DGLP.

5.2 Handling groups and vertex selection

We will use the term *vertex selection* for the choice of which vertex to visit in each group given its assigned agent. Given a group sequence for each agent, the optimal vertex selections may be computed using dynamic programming as described in [25]. Applying a full vertex (re)-selection for every candidate move in a local heuristic is computationally intractable, and therefore some restraint must be taken on where to use a full vertex selection. We will restrict its use to a tandem operation with the even more expensive DGLP. However, faster and cheaper local vertex assignment rules may be applied within the heuristics.

5.3 Handling precedence constraints

Precedence constraints significantly complicate sequencing and balancing heuristics. For intra agent constraints, efficient path preserving k-opt moves have been developed for sequencing heuristics, and path preserving 3-opt which is used here has been particularly successful. Since balancing heuristics such as String Move also are path preserving, these techniques may be generalized and intra agent constraints may be efficiently treated here as well.

Inter agent constraints are much more complicated to handle, and especially the relaxed version. Below, we will define some variables that may give some insight into the effect of a local move on inter agent constraints. These variables are best understood in the context of preventing delays due to inter agent constraints to occur at all. We will therefore make some temporary assumptions on solutions being delay free. For a given solution s, define the *margin* for each precedence constraint $(p, q) \in \Pi$ as

$$M_{p,q} = T_{\text{sta}}(s, q) - T_{\text{end}}(s, p)$$

Assume no inter agent constraint delays exist in the current solution s, that is, for all precedence constraints it holds $M_{p,q} \geq 0$. For a move $s \to s'$, we define the *time shift* for group g as

$$\Delta_g := T_{\text{sta}}(s', g) - T_{\text{sta}}(s, g)$$

For s', any potential delays due to inter agent constraints are ignored, and $T_{\text{sta}}(s', g)$ may thus be calculated just by summing up edge and node times. This definition extends in a natural way even if s already contains delays, which will be clear below when considering the actual heuristics used. In order to prevent introductions of delays due to inter agent constraints, it is clear that this is equivalent to that $\Delta_p - \Delta_q \leq M_{p,q}$ holds for all precedence constraints. For a certain heuristic move, there is a certain maximum and minimum time shift for all groups involved, and by computing once a sorted list of all precedence constraints for s by increasing margins, we only need to check the most critical constraints.

To extend the methodology just described for situations where new delays are introduced, we define the *delay* for the inter agent (w.r.t. s') constraint $(p, q) \in \Pi$ with margin $M_{p,q}$ as

$$d_{p,q} := (\Delta_p - \Delta_q - M_{p,q})_+$$

When estimating the effect of any move, we initially estimate agent dependent tour time gains γ_a as the difference between the removed edge times and added edge times, taking local node reassignments into account but ignoring inter agent constraints. A simple way to incorporate delays is to estimate the new agent tour times by

$$\widehat{T}_{\text{tot}}(s', a) = T_{\text{tot}}(s, a) - \gamma_a + \max_{A(s,q)=a} d_{p,q}$$

Of course, this estimate is very simplified and does not take into account chain effects induced by new waiting times. It is only guaranteed to give correct results when no new delays are introduced, and may not work very well in constraint dense problems. However, some preliminary tests showed that the ten most promising moves almost always contained the actual best move for most test problems. In the most dense problems though, the optimal move could end up outside the top 100 most promising moves.

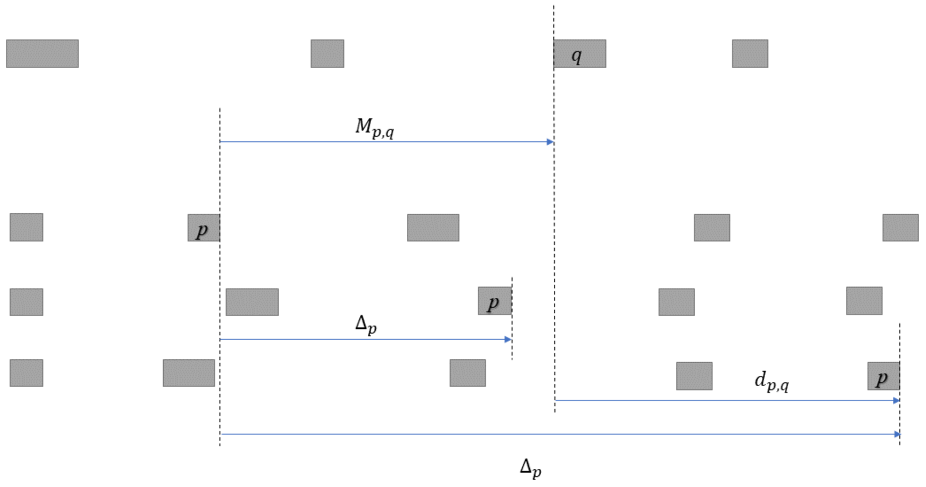


Figure 3: Margin, shifts and delays for two 3-opt moves. The first two lines show parts of two agent tours, the position of groups p and q in the tours, and in between the margin $M_{p,q}$. The third line shows a 3-opt move for the second agent, the new position for group p and the corresponding time shift Δ_p. The fourth line shows another 3-opt move for the second agent, the new position for group p, the corresponding time shift Δ_p and the induced delay $d_{p,q}$.

5.4 Path Preserving 3-opt

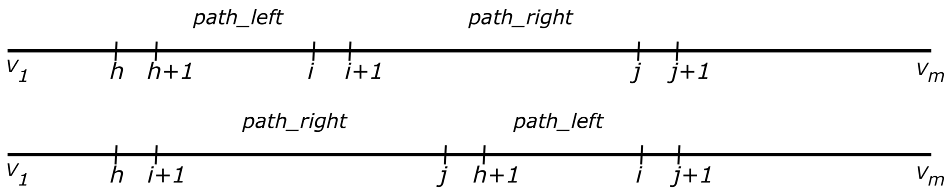


Figure 4: A path preserving 3-opt, forward version.

We apply the Path Preserving 3-opt procedure of Gambardella et al. to each agent tour separately without considering vertex reassignment. Figure 4 illustrates how a right and a left part are identified and swapped in an agent tour. The sequences within the paths are preserved so the only intra agent precedence constraints to consider are those $(p, q) \in \Pi$ with p in the right and q in the left path.

The inter agent constraints may also be affected since the right path is moved to an earlier and the left path to a later point of time. More exactly, given a suggested move (h, i, j) the time shifts for groups in the right and the left segments are

$$\Delta_{\text{ri}} = T_{\text{sta}}(s, a, h) - T_{\text{sta}}(s, a, i+1) + \widetilde{T}_{s^a_h, s^a_{i+1}}$$
$$\Delta_{\text{le}} = T_{\text{sta}}(s, a, i+1) - T_{\text{sta}}(s, a, j) - \widetilde{T}_{s^a_h, s^a_{i+1}} - \widetilde{T}_{s^a_j, s^a_{h+1}} + \widetilde{T}_{s^a_h, s^a_{h+1}}$$

5.5 String Move

Several heuristics aimed at moving tasks (vertices or groups) between agents were introduced in [9] for the VRP. In our setting, the goal is primarily to redistribute groups between agents in solutions where the working load is unbalanced, that is, agent tour times differs significantly. Therfore, we refer to these heuristics as *load balancing heuristics*. From the three heuristics we opted for String Relocation, or String Move as it is often referred to.

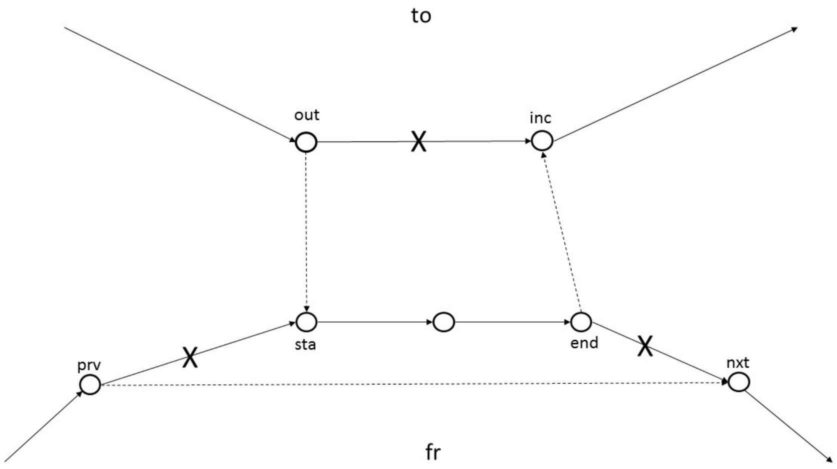


Figure 5: A string move. Edges with an "X" are removed, dashed edges are added.

A string move from agent a_{fr} to agent a_{to} is depicted in Figure 5. It is realized by an outer loop on g_{out} followed by an inner loop over g_{sta}. From there the segment $\sigma = (g_{\text{sta}}, \ldots g_{\text{end}})$ may be built up incrementally by the innermost loop up to some maximum segment length $l_{\max}$. Vertex selection needs to be considered for the groups in the segment σ since they are assigned to a new agent. This is done incrementally in a greedy fashion. When g_{sta} is chosen, the best vertex choice v_{sta} with $G(v_{\text{sta}}) = g_{\text{sta}}$ and $A(v_{\text{sta}}) = a_{\text{to}}$ is made by minimizing

$$\lambda_1 = \widetilde{T}_{v_{\text{out}}, v_{\text{sta}}} + \widetilde{T}_{v_{\text{sta}}, v_{\text{inc}}}$$

For the k:th group g_k to be added to σ, the vertex choice v_k with $G(v_k) = g_k$ and $A(v_k) = a_{\text{to}}$ is made to minimize

$$\lambda_k = \lambda_{k-1} + \widetilde{T}_{v_{k-1}, v_k} + \widetilde{T}_{v_k, v_{\text{inc}}} - \widetilde{T}_{v_{k-1}, v_{\text{inc}}}$$

New potential intra agent constraints violations for agent a_{to} may be introduced between groups in σ and groups from g_{out} and forward. These may be checked by the same lexicographic labeling procedure as for path preserving 3-opt.

New inter agent constraint violations may be introduced between groups in σ and the remaining groups left for agent a_{fr}. Further, due to time shifts, new and changed delays due to inter agent constraints may result. Constraints $(p, q) \in \Pi$ with $A(p) = a_{\text{to}}$ may cause new delays appear in two ways. Assume $T(\sigma, a)$ to be the total processing time of σ when sequenced on agent a. If p is a successor of g_{out} in the old tour, there is a time shift

$$\Delta_p = T(\sigma, a_{\text{to}}) + \widetilde{T}(v_{\text{out}}, v_{\text{sta}}) + \widetilde{T}(v_{\text{end}}, v_{\text{inc}}) - \widetilde{T}(v_{\text{out}}, v_{\text{inc}}).$$

For q in the sequence starting with g_{nxt} the time shift is given as

$$\Delta_q = \widetilde{T}(v_{\text{prv}}, v_{\text{nxt}}) - T(\sigma, a_{\text{fr}}) - \widetilde{T}(v_{\text{prv}}, v_{\text{sta}}) - \widetilde{T}(v_{\text{end}}, v_{\text{nxt}})$$

For p or q in σ similar time shifts are easily calculated.

5.6 Delay Removal

While 3-opt and String Move attempted to avoid introducing delays due to inter agent constraints, they may still occur. Further, it may be the case that the optimal solution contains delays. Still, delays are automatically detected during DGLP, and it could be worthwhile to try to remove them. The following is based on known techniques for reducing idle times in job shop scheduling problems [44].

Assume that we have a solution s with an inter-agent precedence constraint $(g_1, g_2) \in \Pi$ that induces a delay for g_2

$$D(g_2) := T^{\text{sta}}(g_2) - T^{\text{sta}}(g_2') - T^{\text{edge}}_{v_2',v_2}$$

where $G(v_2) = g_2$, $G(v_2') = g_2'$ and (v_2', v_2) is an edge in s.

The idea is that the waiting time for agent a_2 ($A(v_2) = a_2$) may be used for visiting groups later in that agent sequence that are not subject to constraints with g_1 or g_2. A simple way to do this is to move v_2 forward in the sequence while keeping the relative order between the other vertices intact. This is done incrementally one step at a time. In each step, time shifts, gain, and estimated new $T^{\text{sta}}(g_2)$ are easily updated by adding and subtracting edge times in a straightforward manner, and a new estimated agent tour time for a_2. The procedure is repeated for each inter agent constraint as long as there is still a delay. A similar procedure is also employed for each inter agent constraint by moving g_1 backwards in its agent sequence.

A drawback with the suggested procedure is that one often destroys a very good (at least with delays not taken into account) sequence. In most cases, we saw that this meant that the potential gain of using the delay time to visit another group was eliminated by increased edge processing times. But it happened sometimes that an actual improvement was found, so it might still be worthwhile to employ this heuristic.

5.7 Measuring improvement

How to measure the improvement of a particular move in a local search neighbourhood may seem trivial at a first glance; it is simply a decrease in cycle time. But it turns out that such an improvement measure can lead to undesired deadlocks which may be illuminated by some simple hypothetical examples. Consider a three agent problem instance where the optimal solution has individual agent tour times $(10, 10, 10)$. Now suppose that the currently best solution - start solution say - has agent tour times $(5, 12, 12)$. Further assume that no agent tour may be further improved by 3-opt. The cycle time is impossible to decrease in a single string move since one of the agents will always still have 12 as its tour time.

A resolution to this deadlock would be to define the improvement of a string move as the decrease in $\max(T_{\text{tot}}(s, a_{\text{fr}}), T_{\text{tot}}(s, a_{\text{to}}))$. This would allow an improving move to partly balance two of the agents leading to tour times $(8, 10, 12)$ for instance. From there, agent 1 and 3 may be balanced in another improving string move to obtain an optimal solution.

In the same way, it may be reasonable to define the improvement of a 3-opt move to be the largest individual tour time reduction. Consider a hypothetical scenario with two agents and tour times $(12, 12)$. Assume further that 3-opt may lower one tour time but not the other. It is reasonable to consider lowering one of them to say 8 to be an improvement since this may enable a subsequent cycle time improvement via string move.

But since all agent tour times may be affected by any heuristic due to delays, it is preferable to have a general improvement measure that is applied equally to all heuristics. Let $\mathbf{T}(s)$ denote the vector of agent tour times for solution s sorted in descending order. We then define an improvement measure by

$$I(s_{\text{old}}, s_{\text{new}}) = L_K(\mathbf{T}(s_{\text{old}}), \mathbf{T}(s_{\text{new}}))$$

where the "Lexicographic comparison function" L_n is recursively defined by

$$L_n(x, y) = \begin{cases} L_{n-1}(x_{2:n}, y_{2:n}) & \text{if } x_1 = y_1 \text{ and } n > 1 \\ x_1 - y_1 & \text{otherwise.} \end{cases}$$

It is easy to realize that this improvement measure will incorporate the ideas described above for String Move and 3-Opt above and consider a move from agent tour times $(5, 12, 12)$ to $(8, 10, 12)$ an improvement. It will never consider a cycle time increasing move an improvement though, regardless of how small the cycle time increase is and how the other agent tour times are improved. It is possible to relax the improvement function to allow small cycle time increases and still consider a move an improvement, but this will not be explored further here.

6 Results

6.1 Test bed problems

Because of the lack established test problems, there was a need to define new proper test problems. We opted to use the existing SOP instances in TSPLIB and to extend them to PCmGTSP instances according to the following principles

- We create one dummy singleton start group for each agent with all edge times set to zero.
- For each SOP vertex we create a group with $N_{\mathrm{A,G}}$ vertices, that is, $N_{\mathrm{A,G}}K$ vertices in total.
- For a proper edge between two vertices belonging to the same agent a and different groups corresponding to SOP vertices i and j, the corresponding edge time given as

  $$\left\lceil c_{ij}\frac{1+\sigma_a u}{1+\sigma_a/2}\right\rceil$$

  where u is a unit uniform random variable, σ_a are agent-dependent *spread* parameters, and c_{ij} is the SOP edge cost.
- All vertex times are set to zero.

A two-agent version was run for each test problem. For problems with $M \geq 40$, a three-agent version is also run, and for $M \geq 60$ a three-agent version as well. In all cases we used $\sigma_a = a$.

6.2 Algorithm parameters

The following parameter settings were used

- Number of ants: $P = 10$
- Number of ant generations: $R = 100$
- Initial pheromone level: $\tau_0 = 0.5$
- Other HACS parameters: $d_0 = 0.90$, $\alpha = 1$, and $\beta = 2$
- Maximum string length in String Move: $l_{\max} = 5$
- Maximum number of local search iterations: $k_{\max} = 20$
- Maximum number of candidates evaluated by DGLP in each neigbourhood: $j_{\max} = 20$

For the two latter parameters, the pairs (50,50) and (100,100) were also tested except for the larger problems where the total CPU time would be too high.

6.3 Experimental results

Below, in Tables 1, 2, and 3, we present numerical results from our experiments. The values presented in the tables are the mean of the best solution values across the 10 trial runs (Mean), the standard deviation (SD), and the mean running time for the algorithm (Time (s)).

Since this problem has not been considered before and there is a lack of tight lower bounds for the problem instances, the quality of the solutions cannot be accurately assessed. However, some general conclusions may be made from the results. The variation of the best solution value between trials are typically a few percent, and the standard deviation generally seems to decrease as $k_{\max}$ and $j_{\max}$ are increased. Increasing the values of $k_{\max}$ and $j_{\max}$ also seems to improve the solution with about 0-10 % and in some cases even more than that. Increasing from (20,20) to (50,50) pays off the most, particularly for dense scenarios.

The execution times are generally quite high and seem to approximately double with the values on $k_{\max}$ and $j_{\max}$. For the more dense problem instances, the increase seems to be even higher. This is probably a case of the whole candidate solution list being fully analyzed more often since the estimations of improvements are less indicative of the actual improvement in dense instances.

Table 1: Results for instances with 2 agents.

Instance	$k_{\max} = 20$, $j_{\max} = 20$			$k_{\max} = 50$, $j_{\max} = 50$			$k_{\max} = 100$, $j_{\max} = 100$		
	Mean	SD	Time (s)	Mean	SD	Time (s)	Mean	SD	Time (s)
2a.br17.10	14.0	0.0	5.4	14.0	0.0	10.7	14.0	0.0	17.5
2a.br17.12	14.0	0.0	5.2	14.0	0.0	10.8	14.0	0.0	14.3
2a.ESC07	407.0	0.0	1.3	407.0	0.0	1.5	407.0	0.0	1.5
2a.ESC12	444.0	0.0	4.3	444.0	0.0	7.6	444.0	0.0	10.3
2a.ESC25	553.2	12.9	12.1	538.1	15.6	21.1	525.0	16.3	32.8
2a.ESC47	621.5	40.2	45.2	539.0	29.6	92.3	545.3	20.4	131.7
2a.ESC63	31.4	0.5	94.3	30.9	0.3	159.2	31.0	0.0	241.3
2a.ESC78	5160.7	149.6	89.5	4736.7	66.3	276.7	4621.3	72.6	469.4
2a.ft53.1	2799.0	67.1	45.5	2519.0	71.7	119.7	2490.7	19.7	176.2
2a.ft53.2	3201.2	50.3	39.9	2885.6	77.3	117.6	2785.1	82.7	186.5
2a.ft53.3	4353.1	94.9	46.5	3813.3	67.0	161.3	3763.3	79.8	285.5
2a.ft53.4	5185.5	223.7	79.7	4688.2	155.2	250.5	4673.9	91.9	474.5
2a.ft70.1	14862.8	110.1	66.6	13902.2	99.3	151.6	13578.4	141.5	224.4
2a.ft70.2	15577.4	112.1	63.0	14597.7	130.3	148.1	14360.3	88.5	226.1
2a.ft70.3	17022.1	241.3	64.0	16083.0	156.1	180.5	15850.4	147.1	295.5
2a.ft70.4	19681.9	207.5	153.1	18602.2	157.8	515.7	18266.8	249.3	966.4
2a.kro124p.1	21326.9	475.2	201.2	18127.9	331.3	454.5	16101.8	326.6	950.1
2a.kro124p.2	23032.2	710.3	160.4	19441.6	322.2	399.3	17535.0	251.5	914.8
2a.kro124p.3	28417.4	686.9	128.6	24675.0	407.7	419.6	22511.4	507.0	1050.3
2a.kro124p.4	34763.2	1154.8	276.4	30194.0	488.3	1268.3	27305.1	395.5	3214.4
2a.p43.1	1023.2	8.4	24.4	1013.7	15.9	50.7	1012.2	13.6	88.2
2a.p43.2	1303.2	30.7	23.1	1307.9	22.4	59.7	1274.5	24.5	112.2
2a.p43.3	1546.9	36.6	24.7	1539.1	25.8	67.3	1561.8	23.8	133.2
2a.p43.4	2219.3	56.4	49.1	2087.4	153.7	143.1	1825.1	163.6	280.9
2a.prob42	112.9	4.8	30.3	108.4	4.0	50.3	107.7	3.3	73.7
2a.prob100	1212.2	47.2	101.3	845.8	20.8	384.0	786.4	36.4	605.5
2a.rbg048a	128.0	2.4	77.4	121.9	3.8	169.9	121.9	1.5	270.2
2a.rbg050c	166.6	2.2	85.0	160.0	1.7	204.1	159.6	1.7	330.3
2a.rbg109a	435.9	9.5	577.5	384.6	7.8	1644.4	373.8	3.5	2229.9
2a.rbg150a	719.5	7.6	1347.1	643.2	7.2	5153.2	-	-	-
2a.rbg174a	856.3	8.3	1566.2	771.5	8.1	4614.3	-	-	-
2a.rbg253a	1329.2	15.2	4378.9	1188.8	20.4	19456.0	-	-	-
2a.rbg323a	1597.0	19.6	5510.6	-	-	-	-	-	-
2a.rbg341a	1558.7	12.1	8014.1	-	-	-	-	-	-
2a.rbg358a	1597.4	21.9	7762.1	-	-	-	-	-	-
2a.rbg378a	1670.3	31.0	7822.3	-	-	-	-	-	-
2a.ry48p.1	6085.7	171.8	37.0	5276.9	182.5	110.0	5275.7	46.7	174.0
2a.ry48p.2	6551.5	179.7	34.6	5760.6	133.4	110.2	5673.4	137.6	178.1
2a.ry48p.3	7756.2	160.6	35.7	7036.9	138.2	107.0	6903.1	151.9	188.3
2a.ry48p.4	9892.1	248.9	71.3	9098.1	238.9	238.9	9036.8	104.1	428.4

Table 2: Results for instances with 3 agents.

Instance	$k_{\max} = 20$, $j_{\max} = 20$			$k_{\max} = 50$, $j_{\max} = 50$			$k_{\max} = 100$, $j_{\max} = 100$		
	Mean	SD	Time (s)	Mean	SD	Time (s)	Mean	SD	Time (s)
3a.ESC47	357.8	28.4	41.0	300.4	17.6	121.2	300.6	8.0	179.2
3a.ESC63	20.9	0.3	86.9	20.0	0.0	174.9	20.0	0.0	262.4
3a.ESC78	3000.3	111.4	112.2	2598.0	78.0	346.5	2468.5	86.9	719.0
3a.ft53.1	1663.1	37.8	41.7	1394.3	25.2	132.0	1344.1	16.7	251.8
3a.ft53.2	1860.8	37.1	45.6	1618.9	31.3	130.6	1531.9	29.4	262.3
3a.ft53.3	2654.9	81.0	53.5	2149.7	82.2	191.4	2008.5	46.9	391.9
3a.ft53.4	3216.9	183.2	96.1	2758.7	49.2	400.1	2610.2	45.3	748.0
3a.ft70.1	8337.3	145.1	61.5	7961.2	93.1	171.9	7789.4	54.1	388.9
3a.ft70.2	8723.3	163.6	62.3	8344.1	80.7	172.9	8091.0	71.8	405.7
3a.ft70.3	9795.8	197.8	73.3	9159.0	130.2	231.3	8845.7	108.6	519.7
3a.ft70.4	12210.3	243.0	179.6	11013.6	150.2	680.4	10646.7	69.6	1431.3
3a.kro124p.1	12355.8	538.1	170.7	10350.9	156.3	413.9	9159.2	186.1	1056.4
3a.kro124p.2	13466.0	449.9	158.7	10964.4	356.1	391.1	9745.7	209.5	997.3
3a.kro124p.3	17491.7	480.2	172.3	14382.4	455.2	471.3	12698.9	117.3	1116.7
3a.kro124p.4	22459.5	813.4	369.1	18585.7	480.7	1267.5	16442.5	287.6	3292.2
3a.p43.1	302.2	11.2	33.4	272.7	14.7	64.4	259.8	7.5	113.5
3a.p43.2	333.7	12.3	33.2	305.4	7.7	74.8	291.9	6.33	137.3
3a.p43.3	413.1	19.4	34.9	394.1	8.3	91.6	386.9	10.7	185.0
3a.p43.4	678.1	15.6	74.6	627.8	13.8	202.3	610.1	13.00	321.7
3a.prob42	65.8	2.9	31.7	62.5	1.8	70.3	60.8	2.2	104.6
3a.prob100	808.5	50.8	128.5	495.7	22.7	414.9	452.9	13.7	884.3
3a.rbg048a	71.1	1.5	73.5	66.8	1.7	170.4	64.5	1.3	304.4
3a.rbg050c	93.0	1.1	84.6	86.6	1.3	221.7	85.7	1.4	383.7
3a.rbg109a	263.8	8.6	643.9	-	-	-	-	-	-
3a.rbg150a	452.4	14.8	1462.8	-	-	-	-	-	-
3a.rbg174a	533.2	11.4	2020.6	-	-	-	-	-	-
3a.rbg253a†	-	-	-	-	-	-	-	-	-
3a.rbg323a†	-	-	-	-	-	-	-	-	-
3a.rbg341a†	-	-	-	-	-	-	-	-	-
3a.rbg358a†	-	-	-	-	-	-	-	-	-
3a.rbg378a†	-	-	-	-	-	-	-	-	-
3a.ry48p.1	3468.3	101.2	39.5	3033.5	57.7	124.6	2927.4	47.4	225.1
3a.ry48p.2	3721.6	100.0	38.7	3240.0	89.8	119.8	3127.7	40.7	236.7
3a.ry48p.3	4410.4	184.8	47.1	3900.0	117.6	136.7	3738.5	104.1	262.9
3a.ry48p.4	5699.7	148.5	88.2	5063.9	165.6	286.0	4894.4	163.7	539.1

† Execution times were very high and the instance was therefore skipped.

Table 3: Results for instances with 4 agents.

Instance	$k_{\max} = 20$, $j_{\max} = 20$			$k_{\max} = 50$, $j_{\max} = 50$			$k_{\max} = 100$, $j_{\max} = 100$		
	Mean	SD	Time (s)	Mean	SD	Time (s)	Mean	SD	Time (s)
4a.ESC63	16.0	0.0	105.5	14.9	0.3	211.1	15.0	0.0	334.6
4a.ESC78	2157.6	66.8	142.5	1844.3	57.2	377.4	1723.9	52.2	759.8
4a.ft70.1	6200.7	97.5	80.2	5796.6	63.8	209.4	5642.4	52.1	445.2
4a.ft70.2	6674.3	119.5	84.4	6129.2	109.7	226.1	5943.0	58.5	511.4
4a.ft70.3	7596.2	201.7	97.9	6775.4	84.1	286.7	6436.0	95.1	670.3
4a.ft70.4	9517.8	257.0	218.2	8293.4	147.4	819.0	7779.6	169.1	1918.6
4a.kro124p.1	9546.3	379.3	206.7	-	-	-	-	-	-
4a.kro124p.2	10230.0	408.1	201.8	-	-	-	-	-	-
4a.kro124p.3	12614.0	824.2	227.6	-	-	-	-	-	-
4a.kro124p.4	16493.0	551.3	468.5	-	-	-	-	-	-
4a.prob100	521.8	28.6	197.0	-	-	-	-	-	-
4a.rbg109a	212.6	10.4	781.7	-	-	-	-	-	-
4a.rbg150a	363.0	5.1	1738.5	-	-	-	-	-	-
4a.rbg174a	452.6	4.8	2338.9	-	-	-	-	-	-
4a.rbg253a	715.5	15.8	11451.0	-	-	-	-	-	-
4a.rbg323a	869.3	25.5	27086.0	-	-	-	-	-	-
4a.rbg341a	825.4	13.3	30213.0	-	-	-	-	-	-
4a.rbg358a†	-	-	-	-	-	-	-	-	-
4a.rbg378a†	-	-	-	-	-	-	-	-	-

† Execution times were very high and the instance was therefore skipped.

7 Conclusions and future work

In this paper we have described the hitherto fairly unexplored problem PCGmTSP. To match the underlying applications prompting this investigation we considered the variant with minmax objective, and where precedence constraints between agents are included and interpreted in the soft sense so that they may be violated causing delays. This complicates the algorithms and also renders objective function evaluation relatively expensive. We developed some techniques for handling inter agent precedence constraints by letting some cheaply calculated estimates of agent tour times guide the full solution evaluation to a few number of promising candidate solutions. These techniques were incorporated together with local node assignment procedures into the already established local search heuristics path preserving 3-opt, String Move and Delay Removal. These heuristics were then included in a Hybrid Ant Colony Search (HACS) algorithm which was tested on a series of test bed problems developed by adding groups structure to standard SOP instances. The preliminary conclusions to be drawn from these experiments are that this is indeed a very difficult problem which requires a heavy computational load to reach good solutions.

There are many possible paths for further investigations into this complicated problem, and we will listen a few below which have attracted our main interest. To start with, other meta heuristics could be tried as well. We did some preliminary tests with a pretty standard tabu search method, but it performed significantly worse than HACS so we did not explore it further. We conjecture that it needs to be augmented with some more elaborate long time memory structures for better sampling and exploration of the solution space. A genetic algorithm may also be well suited to this kind of problem where objective value computation is relatively expensive, especially for dense precedence constraints where it is difficult to predict the outcome of a local change.

Since the main bottleneck of the algorithm is objective function evaluation, an obvious line of further inquiry is to find more elaborate and efficient ways to reduce the number of such function calls. A first simple step could be to estimate the average number of candidate solutions evaluated. If this number is smaller than the candidate list size, the latter could be decreased and vice versa. For really dense instances, one could question if local search should be applied at all. Perhaps a pure genetic algorithms would be better in this case.

In order to better assess the solution quality of heuristics, methods for obtaining an optimum or a lower bound on the optimum may be interesting.

References

[1] P. Toth and D. Vigo, *The Vehicle Routing Problem*, Society for Industrial and Applied Mathematics (2002).

[2] B. Golden, S. Raghavan, E. Wasil, *The Vehicle Routing Problem: Latest Advances and New Challenges*, Springer (2008).

[3] B.L. Golden, G. Laporte, E.D. Taillard, *An adaptive memory heuristic for a class of vehicle routing problems with minmax objective*, Computers & Operations Research 24(5) (1997), pp. 445-452.

[4] S. Somhom, A. Modares, T. Enkawa, *Competition-based neural network for the multiple travelling salesmen problem with minmax objective*, Computers & Operations Research 26 (1999), pp. 395—407.

[5] S. Yuan, B. Skinner, S. Huang, D. Liu, *A new crossover approach for solving the multiple travelling salesmen problem using genetic algorithms*, European Journal of Operational Research 228 (2013), pp. 72-82.

[6] P. Venkatesh and A. Singh, *Two metaheuristic approaches for the multiple traveling salesperson problem*, Applied Soft Computing 26 (2015), pp. 74-89.

[7] B. Soylu, *A general variable neighborhood search heuristic for multiple traveling salesmen problem*, Computers & Industrial Engineering 90 (2015), pp. 390-401.

[8] Y. Wang, Y. Chen, Y. Lin, *Memetic algorithm based on sequential variable neighborhood descent for the minmax multiple traveling salesman problem*, Computers & Industrial Engineering 106 (2017), pp. 105-122.

[9] A. van Breedam, *Improvement Heuristics for the Vehicle Routing Problem based on Simulated Annealing*, European Journal of Operational Research 86 (1995), pp. 480-490.

[10] E. Balas, *Machine Sequencing via Disjunctive Graphs: An Implicit Enumeration Algorithm*, Operations Research 17(6) (1969), pp. 941-957.

[11] A. Allahverdi, C.T. Ng, T.C.E. Cheng, Mikhail Y. Kovalyov, *A survey of scheduling problems with setup times or costs*, European Journal of Operational Research 187 (208), pp. 985-1032.

[12] A. Allahverdi, *The third comprehensive survey on scheduling problems with setup times/costs*, European Journal of Operational Research 246 (215), pp. 345-378.

[13] L.F. Escudero *A inexact algorithm for the sequential ordering problem*, European Journal of Operational Research 37(2) (1988), pp. 236-249.

[14] 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.

[15] A.A. Ciré and WJ. van Hoeve, *Multivalued Decision Diagrams for Sequencing Problems*, Lecture Notes in Computer Science vol 8656 (2014).

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

[17] 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.

[18] 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.

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

[20] S.S. Srivastava, S. Kumar, R.C. Garg, P. Sen, *Generalized travelling salesman problem through n sets of nodes*, CORS Journal 7(2) (1969), pp. 97-101.

[21] G. Laporte and Y. Nobert, *Generalized travelling salesman problem through n sets of nodes: an integer programming approach*, INFOR: Information Systems and Operational Research 21(1) (1983), pp. 61-75.

[22] G. Laporte, H. Mercure, Y. Nobert, *Finding the shortest Hamiltonian circuit through n-clusters*, Congressus Numerantium (1985), pp. 277-290.

[23] 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.

[24] 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.

[25] 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.

[26] 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.

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

[28] 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.

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

[30] 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.

[31] 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.

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

[33] 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.

[34] 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.

[35] 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.

[36] 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.

[37] V. Kachitvichyanukul, P. Sombuntham, S. Kunnapapdeelert, *Two solution representations for solving multi-depot vehicle routing problem with multiple pickup and delivery requests via PSO*, Computers & Industrial Engineering 89 (2015), pp. 125-136.

[38] A. Goel and V. Gruhn, *A General Vehicle Routing Problem*, European Journal of Operational Research 191 (2008), pp. 650-660.

[39] W. Malik, S. Rathinam, S. Darbha, *An approximation algorithm for a symmetric Generalized Multiple Depot, Multiple Travelling Salesman Problem*, Operations Research Letters 35 (2007), pp. 747–753.

[40] D. Spensieri, J.S. Carlson, F. Ekstedt, R. Bohlin, *An Iterative Approach for Collision Free Routing and Scheduling in Multirobot Stations*, IEEE Transactions on Automation Science and Engineering 13(2) (2016), pp. 950-962.

[41] M. Afzalirad and J. Rezaeian, *Resource-constrained unrelated parallel machine scheduling problem with sequence dependent setup times, precedence constraints and machine eligibility restrictions*, Computers & Industrial Engineering 98 (2016), pp. 40-52.

[42] S.C. Sarin, H.D. Sherali, J.D. Judd, P-F. Tsai, *Multiple asymmetric traveling salesmen problem with and without precedence constraints: Performance comparison of alternative formulations*, Computers & Operations Research 51 (2014), pp. 64-89.

[43] A.H. Gharehgozli, G. Laporte, Y. Yu, R. de Koster, *Scheduling Twin Yard Cranes in a Container Block*, Transportation Science 49(3) (2017), pp. 685-705.

[44] J. Blazewicz, W. Domschkeb, E. Pesch, *The job shop scheduling problem: Conventional and new solution techniques*, European Journal of Operational Research 93(1) (1996), pp. 1-33.