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 Euclidean distances between tasks. The sequence is then sent to the path plan-

ner which checks how the robots should move in order to feasibly execute it. 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.

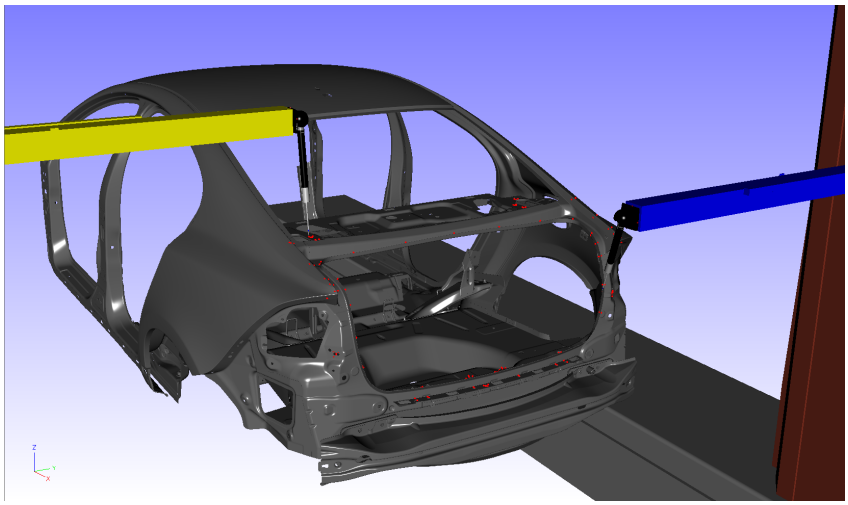


Figure 1.1: Simulation in IPS of a station with two coordinate measuring machines working on a car body.

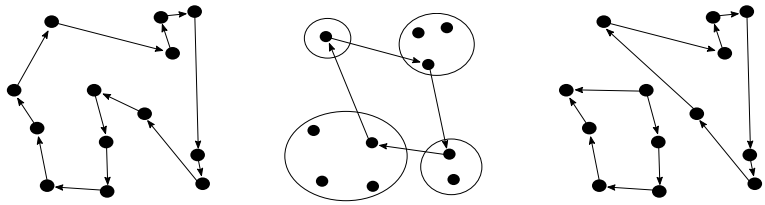
The distances between tasks may become several times as long after finding feasible robot paths, and generally the path planning affects the cycle time much more than the task sequencing. Additionally, optimizing the sequence of tasks with an incomplete set of properly planned paths is in itself an approximation 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.

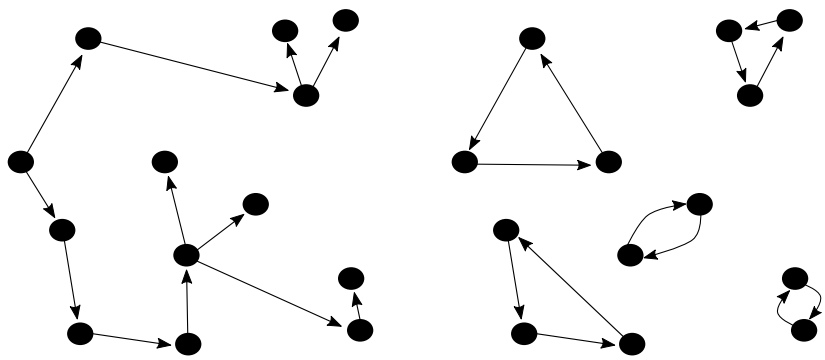
While the PCGTSP and the PCGmTSP can be seen as aggregates of other well-studied problems, they themselves pose their own set of challenges as the methodologies for tackling the different related problems tend to clash. As an example, many effective GTSP heuristic methods rely on the fact that the edges 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 prob-

lems. 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 $V_1, \ldots, V_m$ be a partition of V, i.e. a family of subsets of V such that $V_p \cap V_q = \emptyset$ for $p, q = 1, \ldots, m$, $p \neq q$, and $\bigcup_{p=1}^{m} V_p = V$. Each subset in the partition is called a *group*. Let $M = \{1, \ldots, m\}$ 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, $V_1, \ldots, V_m$, and group index set $M = \{1, \ldots, m\}$.

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.

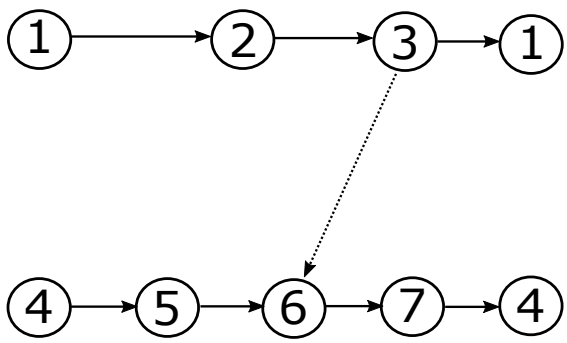


Figure 2.1: Disjunctive graph representation of a two agent solution. Disjunctive edges are dotted.

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.

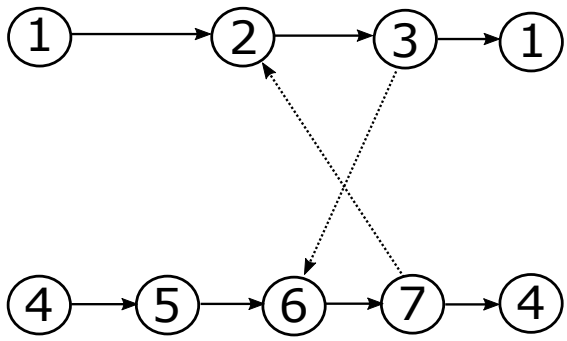


Figure 2.2: Disjunctive graph representation of an infeasible solution. Disjunctive edges are dotted.

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

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

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).

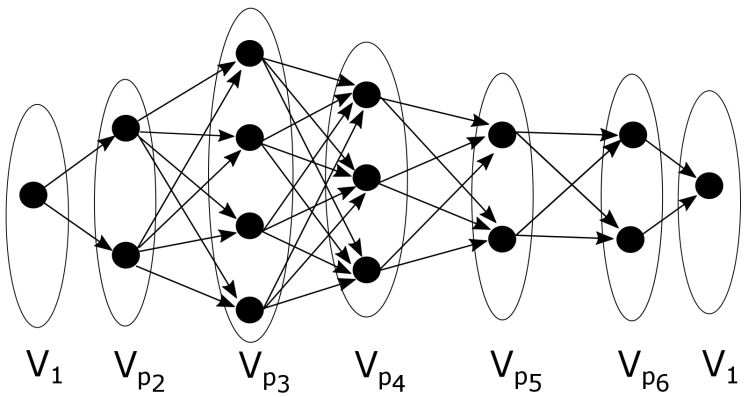


Figure 3.1: A layered network representation of a feasible group sequence. Shortest path through the network gives an optimal vertex selection.

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.

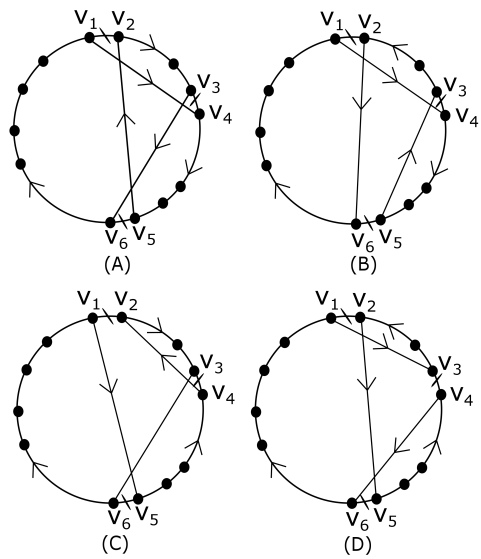


Figure 3.2: The four different types of 3-exchange when edges (v_1, v_2), (v_3, v_4), and (v_5, v_6) are removed. (A) is path preserving.

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 $g(v_i)$, $i = 1, \ldots, 6$, 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 $\sigma = (V_1, \ldots, V_{p_r})$. 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 $\sigma = (V_1, \ldots, V_{p_r})$, 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 $c_{\min}(\sigma)$ 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:

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

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 $c^{(\mathrm{NB})}_{\mathrm{off}}$. 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:

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

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* $\mathrm{NB}(\mathcal{P})$ *denote the ATSP instance which arises when applying the Noon-Bean transformation [40] to the weak version of $\mathcal{P}$.*

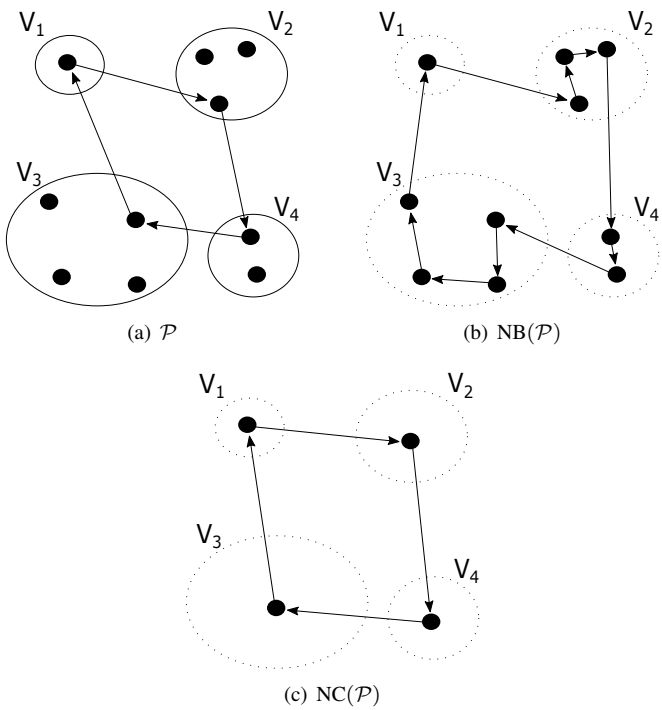
The second approach involves relaxing the vertex choice constraints which enforce the rule that one must enter and exit the same vertex when visiting a group. It has been shown in [39] that by doing this one may define each group

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* 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) NB($\mathcal{P}$)

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

Figure 3.3: Feasible tours in a GTSP instance $\mathcal{P}$ and the transformations NB($\mathcal{P}$) and 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 NC($\mathcal{P}$) is always smaller than or equal to the size of 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 NC($\mathcal{P}$) is defined on a smaller graph than NB($\mathcal{P}$), it is a potentially weaker formulation. Furthermore, 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 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* 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 $k = k_{\text{max}}$ 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 $k = 1, \ldots, |N(\tilde{s})|$.
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 $k = j_{\max}$, 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).

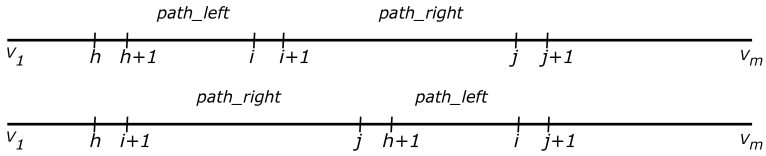


Figure 4.1: A path preserving 3-exchange for a tour visualized as a path. Top path represents the tour before the exchange, and the bottom one after the exchange.

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.

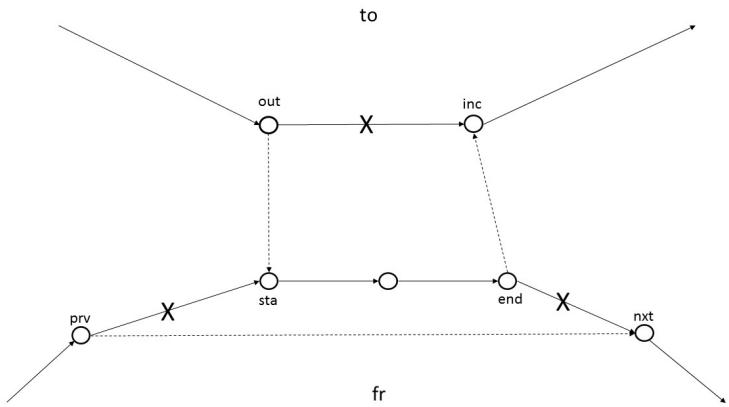


Figure 4.2: A string move. Edges with an "X" over them are removed while dashed edges are added.

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 fea-

sible 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.