Intermittent traveling salesman problem

Tullemans, Remon H.

Award date:
2019

EINDHOVEN UNIVERSITY OF TECHNOLOGY

Bachelor Thesis in Combinatorial Optimization

Intermittent Traveling Salesman Problem

by Remon Tullemans

Abstract This paper analyses the Intermittent Traveling Salesman Problem (ITSP), a variant of the well-known Traveling Salesman Problem (TSP). Special cases, where distances are zero or distances are substantially large, are discussed and lower bounds on the optimal completion time are given. Moreover, two models are proposed to solve the ITSP. The first model gives an optimal solution for cases where nodes heat up and cool down linearly, but cooling down can happen slower. The second model gives a feasible solution, close to an optimal one, when all distances are equal to zero.

2WH40
Bachelor Thesis (10 credits ECTS)
June 2019
Supervisor: prof. dr. Frits C.R. Spieksma

Contents

1 Introduction

In nowadays mathematics, the Traveling Salesman Problem (TSP) is a broadly known problem that has been studied since the mid 1800's [1]. The problem is to find a routing of a salesman who starts from a home location, visits a prescribed set of cities and returns to the original location in such a way that the total distance travelled is minimum and each city is visited exactly once [2]. Of course, minimizing the distance can be replaced by minimizing time, costs, etc. depending on the application. It has been proven that the TSP is NP hard [3].

Multiple variations of the TSP are known, all having different purposes and applications. The one this paper will focus at, is the Intermittent Traveling Salesman Problem (ITSP), where it is better not to talk about cities, but about nodes instead. In the ITSP, each node has to be processed for a given amount of time. Additionally, processing a node causes it to heat up, and eventually melts when its temperature becomes too high. Therefore, the processing must be divided over time intervals, with time lapses for cooling down of the nodes in between [4]. This can be done either by waiting at this node or processing another node.

To actually visualise the problem, the ITSP can be explained by the following example: a salesman gets a list of locations, where at each location, a hole must be drilled for a known amount of time. Drilling a hole causes the temperature to locally rise. Moreover, an upper bound is known specifying the maximum allowable temperature, such that at no point in time at no location, the temperature should exceed this maximum. If this upper bound is reached during the drilling of a hole, drilling of that hole must be stopped in order to allow a cooling down (and of course, other locations can then be visited). The goal is to find a route visiting each location, so that the processing time requirement of each location is fulfilled, and the temperature never exceeds the maximum, while minimizing total completion time.

1.1 Literature research

As already mentioned, multiple variations of the TSP are known, with some of them being quite similar to the ITSP. Their features and acknowledgements can be useful for analysing the ITSP. The most important ones, and their difference to the ITSP, are outlined below:

- TSP with multiple visits (TSPM): The TSPM is a variant of the normal TSP where it can happen that nodes have to be visited more than once, such that for each node u, $v_u \geq 1$ corresponds to the amount of visits needed at node u. The goal is to visit all nodes exactly for the required v_u amount of times with the total distance being minimized. The difference to the ITSP, is that there is no time constraint between two visits. In addition, there is no processing time at each node needed in the TSPM. Nevertheless, some features of the TSPM can be applicable to the ITSP, since multiple visits are allowed in the ITSP. The paper of E. Wacholder studies a neural network to solve the TSPM [5].
- TSP with time windows: In this problem, for each node u there is an interval $[a_u, b_u]$ and a service time s_u, such that node u needs to be fully processed for s_u time units in this interval. The salesman has to wait whenever it arrives before a_u. The difference between the TSP with time windows and the ITSP, is that in this problem, multiple visits at nodes are not considered. Therefore, there is no need to consider temperature of nodes as well, which is in contrast with the ITSP. A detailed study of the TSP with time windows can be found in the paper of Y. Dumas et al. [6] or N. Ascheuer et al. [7].

2 Problem definition

In this section, the ITSP is outlined in mathematical formulation, including all relevant parameters. This will form the base of the rest of this paper. Furthermore, a basic example shows how the ITSP can be visualized. Moreover, some definitions that are used in the rest of the paper are given.

2.1 Problem definition

Let $G = (V, E)$ be a complete, weighted graph with $|V| = n$ vertices. For each vertex $u \in V$, a value $a_u > 0$ corresponds to the minimum time to be spent at node u, which is called the *drilling time* of node u. Moreover, for every edge $(u, v) \in E$ with $u, v \in V$, d_{uv} corresponds to the value of the edge between u and v and represents the time to travel between node u and node v. It is assumed that the triangle inequality holds, such that $d_{uw} + d_{wv} \geq d_{uv}$ $\forall u, v, w \in V$. When a node is being processed, a heating function $f^H(t)$ is given to formalize in what way nodes heat up (linear, exponentially, etc.). Analogously, there is a cooling function $f^C(t)$ to represent in which way nodes are cooling down after they have been processed. $f^H(t)$ and $f^C(t)$ together are called the *heating functions* of the ITSP.

Moreover, there is a global bound $B > 0$, such that the temperature at each node may not exceed B at any moment in time. Hence, a node u may have to be visited multiple times, or an appropriate amount of waiting time is needed to make sure that its temperature does not become too high. All in all, during the complete process, the temperature of a node may never exceed B. The goal is to find an optimal route from a starting point S back to itself where the total time spent is minimal. As a remark, the starting node is a 'dummy node', and thus does *not* have a drilling time.

In addition, we assume that all a_u, d_{uv} and B are integer valued. Furthermore, the process starts at $t = 0$ and all temperatures of all nodes equal zero at $t = 0$. The time needed to get all nodes processed for the required drilling time, is called the *maximum completion time*. The following example illustrates how the ITSP can be visualized.

Example 2.1. *Let a graph with $n = 3$ nodes be given: a starting node S, a red node u and a blue node v. Let $a_u = 9$ and $a_v = 5$, with $B = 4$ as the maximum allowed temperature of a node. The distance between the nodes u and v equals 2, and the distance between S and u equals 0, even as the distance between S and v. As a result, no time is needed to travel from S to another node, such that the first node we want to process can be either u or v. Moreover, the temperature functions are given by $f^H(t) = f^C(t) = t$, such that processing a node for t time units causes the temperature of that node to rise with t (see Figure 1). The following method can be used to process both nodes: start by processing node u for 4 time units, travel to node v, which takes 2 time units, and process node v for 4 time units as well. Afterwards, consecutively process the other node, all for the maximum allowed time. Note that this may not result in the optimal way to process both nodes. It can be seen that this never causes the temperature of one of the nodes above 4 in Figure 2. After 22 time units, both nodes have been processed, and after 23 time units, all nodes have been fully cooled down.*

In figure 2, we see that all nodes have fully been processed after 22 time units, and all nodes have been fully cooled down after 23 time units. We say that the process is done after 22 time units, since none of the nodes needs to be processed after 22 time units. Consequently, we call the maximum completion time of this process 22 time units.

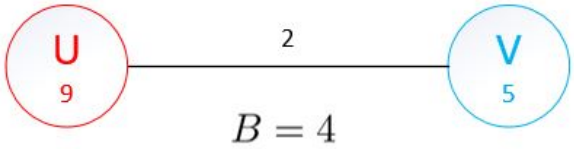


Figure 1: An example graph with 2 nodes, where $B = 4$ is seen as the maximum allowed temperature of a node.

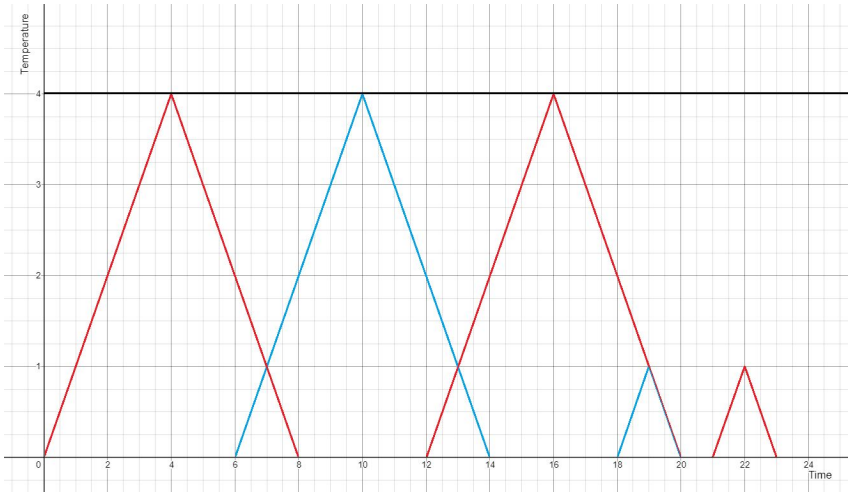


Figure 2: A scheme to process both nodes within 22 time units.

2.2 Definitions and notation

We can see an optimal solution to the ITSP as an order in which nodes have to be processed consecutively, alongside with the processing and waiting time for each node in that order. This order tells us exactly which node has to be processed in which step, and tells us how long this node has to be processed as well. One step of this order is referred to as a *processing step*, notated with p_{us}. So $p_{us} = c$ means that in step s, node u is processed for c time units. Note that s in this case means one step, and not one time unit.

Moreover, to get feasible outcomes, both $f^H(t)$ and $f^C(t)$ are assumed to be continuous and monotonically increasing for $t > 0$ with $f^H(0) = f^C(0) = 0$. If $f^H(t)$ is not monotonically increasing (so decreasing at some interval $[t_1, t_2]$ with $0 < t_1 < t_2$), heating up would essentially mean cooling down at this interval $[t_1, t_2]$, which is undesired. Analogously, cooling down would mean heating up between $[t_1, t_2]$ if $f^C(t)$ is not monotonically increasing. Consequently, if nothing is mentioned, $f^H(t)$ and $f^C(t)$ are assumed to be continuous functions that are monotonically increasing with $f^H(0) = f^C(0) = 0$ throughout this report.

As a remark, it might be strange that $f^C(t)$ has to be monotonically increasing, since it represents cooling down of a node. $f^C(t)$ has to be monotonically increasing because of the following: a formulation is needed what the temperature of a node u is at any time. Therefore, let t be the current time, t_b be the time at the beginning of a processing step, and t_e be the time at the end of a processing step. Furthermore, let $T_u(t)$ be the temperature of a node u at time t. This being said, the temperature of a node u changes as $T_u(t) = T_u(t_b) + f^H(t - t_b)$ when it is being processed. Similarly, its temperature changes as $T_u(t) = \max\{0, T_u(t_e) - f^C(t - t_e)\}$ when u is *not* being processed. We see that $f^C(t)$ needs to be monotonically increasing because of this minus sign in the second expression.

Lastly, some other short definitions are given below:

- $a_{u^*} := \max_u a_u$, so that a_{u^*} is the maximum among all a_u.
- l_{us} is the processing time that is left for a node u at the beginning of time step s.

3 Special cases of the ITSP

In this section, three special cases of the ITSP are discussed. In section 3.1, only one single node is considered and the time needed to get this node fully processed is discussed. In section 3.2, graphs are considered in which the distances between all nodes equal zero. Hence, only processing times and waiting times influence the maximum completion time. Conditions such that no waiting times are needed to process all nodes are given. If these conditions do *not* hold, waiting times need to occur. The maximum completion time in these cases will be given as well. Last, the other extreme is analysed in section 3.3, where distances are very big compared to the drilling times. It is discussed how big the distances need to be such that a TSP tour with processing each node in one visit (either with waiting times) is the optimal way to process all nodes.

Two different pairs of heating functions are analysed in this paper. First, the heating functions are assumed to be the same, meaning that $f^H(t) \equiv f^C(t) := f(t)$. In this case, when $0 < t \leq f^{-1}(B)$ time units are processed at a certain node, it is known that this node has fully cooled down after t time units as well, regardless of what kind of function $f(t)$ exactly is. Second, other features arise when nodes cool down slower than they heat up. Hence, the specific case is investigated where $f^H(t) = t$ and $f^C(t) = \alpha t$ $(0 < \alpha < 1)$. The constant α is in this case called the *cooling down factor*.

3.1 Single node

When a graph consists of only one node u, such that $n = 1$, the goal in the ITSP is to process this node with the least amount of waiting times, such that the temperature of this node does not exceed B. This is due to the fact that the required amount of drilling time a_u needs to be met, and distances have no influence on the maximum completion time. It is not hard to see that it depends on the temperature functions and the value of B how much waiting time (if any) is required.

Therefore, two lemmas are stated and proven to compute how much time is needed to fully process node u as fast as possible. The first lemma assumes that both heating up and cooling down are the same, whereas the second lemma assumes that both functions are linear, while cooling down happens slower.

Lemma 3.1. *Let $G = (V, E)$ be a graph consisting of one node u. Moreover, let $B > 0$, $a_u > 0$ and $f^H(t) \equiv f^C(t) := f(t)$. Then the optimal way to process node u takes*

$$OPT = \max\{a_u,\ 2a_u - f^{-1}(B)\}$$

time units.

Proof. Two cases can be distinguished: $a_u \leq f^{-1}(B)$ or $a_u > f^{-1}(B)$:

1. $a_u \leq f^{-1}(B)$: in this case, node u can be finished by processing it for a_u time units in one step, since $f(a_u) \leq f(f^{-1}(B)) = B$. It can now be seen that $2a_u - f^{-1}(B) = a_u + a_u - f^{-1}(B) \leq a_u$, which results in $\max\{a_u, 2a_u - f^{-1}(B)\} = a_u$ as desired.

2. $a_u > f^{-1}(B)$: consider the following procedure to process node u: start by processing node u first for $f^{-1}(B)$ time units and let $\psi := \lfloor \frac{a_u}{f^{-1}(B)} \rfloor \geq 1$, such that ψ is the amount of times $f^{-1}(B)$ 'fits' into a_u. Then, one can alternately wait and process for $f^{-1}(B)$ time units exactly $\psi - 1$ times, which each take $(\psi - 1)f^{-1}(B)$ time units. Hence, $2(\psi - 1)f^{-1}(B)$ time units are needed in total to get this done. In the end, $0 \leq a_u - \psi f^{-1}(B) < f^{-1}(B)$ time units are left to be processed and the temperature of u equals B, so u can be fully processed by waiting and processing for this amount of time units, which then takes $2(a_u - \psi f^{-1}(B))$ time units. All in all, the maximum completion time equals

   $$f^{-1}(B) + 2(\psi - 1)f^{-1}(B) + 2(a_u - \psi f^{-1}(B)) = 2a_u - f^{-1}(B)$$

   time units. u cannot be processed faster, since it has to be processed for a_u time units and at least $a_u - f^{-1}(B)$ time units are needed to ensure that its temperature does not become too high, resulting in a total time being at least $2a_u - f^{-1}(B)$. Now, it can be seen that $2a_u - f^{-1}(B) = a_u + a_u - f^{-1}(B) > a_u$, which results in $\max\{a_u, 2a_u - B\} = 2a_u - f^{-1}(B)$ as desired.

□

This lemma turns out to be very useful for graphs with more than one node as well, assuming that the temperature functions are the same. This is due to the fact that in bigger graphs, nodes can be processed in one go, which means that they are fully processed without visiting other nodes in the mean time. In this case, this lemma can be used to know what time this takes.

On the other hand, if heating up and cooling down are *not* the same, Lemma 3.1 does not hold. Only the case where $f^H(t) = t$ and $f^C(t) = \alpha t$ $(0 < \alpha < 1)$ is studied, so a similar lemma is stated and proven what time it takes to process exactly one node with these temperature functions. This lemma and its proof are quite similar to Lemma 3.1.

Lemma 3.2. *Let $G = (V, E)$ be a graph consisting of one node u. Moreover, let $B > 0$, $a_u > 0$ and $f^C(t) = \alpha t$ $(0 < \alpha < 1)$. Then the optimal way to process node u takes*

$$OPT = \max\left\{a_u,\ a_u + \frac{a_u - B}{\alpha}\right\}$$

time units.

Proof. Again, two cases can be distinguished: $a_u \leq B$ or $a_u > B$:

1. $a_u \leq B$: in this case, node u can be finished by processing it for a_u time units in one step, since $f^H(a_u) = a_u \leq B$. It can now be seen that $a_u + \frac{a_u - B}{\alpha} \leq a_u$ since $a_u - B \leq 0$ and $\alpha > 0$, which results in $\max\{a_u, a_u + \frac{a_u - B}{\alpha}\} = a_u$ as desired.

2. $a_u > B$: let a similar procedure to the one in Lemma 3.1 be described to process node u: start by processing node u first for B time units and let $\psi := \lfloor \frac{a_u}{B} \rfloor \geq 1$. Then, one can alternately wait for $\frac{B}{\alpha}$ and process for B time units exactly $\psi - 1$ times, which takes a total of $(\psi - 1)(B + \frac{B}{\alpha})$ time units. In the end, $0 \leq a_u - \psi B < B$ is left to be processed and the temperature of u equals B, so u can be fully processed by first waiting for $\frac{a_u - \psi B}{\alpha}$ and subsequently processing for $a_u - \psi B$ amount of time units. As a result, the maximum completion time equals

$$\begin{aligned} B + (\psi - 1)(B + \tfrac{B}{\alpha}) + \tfrac{a_u - \psi B}{\alpha} + a_u - \psi B &= B + \psi B - B + \tfrac{\psi B}{\alpha} - \tfrac{B}{\alpha} + \tfrac{a_u}{\alpha} - \tfrac{\psi B}{\alpha} + a_u - \psi B \\ &= a_u + \frac{a_u - B}{\alpha} \end{aligned}$$

   time units. u cannot be processed faster, since it has to be processed for a_u time units at least $\frac{a_u - B}{\alpha}$ time units are needed to ensure that its temperature does not become too high, resulting in a total time being at least $a_u + \frac{a_u - B}{\alpha}$. Now, it can be seen that $a_u + \frac{a_u - B}{\alpha} > a_u$ since $a_u - B > 0$, which results in $\max\{a_u, a_u + \frac{a_u - B}{\alpha}\} = a_u + \frac{a_u - B}{\alpha}$ as desired.

□

As a result, with the two different pairs of temperature functions considered, it is known what optimal time it takes to process exactly one node. As already mentioned, these lemmas can be useful for graphs with $n > 1$ vertices as well. Therefore, graphs on $n > 1$ vertices are considered next, with the additional assumption that there are no distances between all nodes.

3.2 Zero distances

In the case that distances are zero between all nodes, only drilling times and waiting times influence the maximum completion time. First of all, lower bounds on the maximum total completion time are given for both pairs of temperature functions. When the temperature functions are the same, the next lemma provides a lower bound on the optimal solution. On the other hand, when $f^H(t) = t$ and $f^C(t) = \alpha t$ (0¡α¡1), a corollary of this theorem can be used to state what the maximum total completion time is in this case.

The ITSP considers a starting node S, where the whole process needs to start and end. Note that because the distances between all nodes are equal to zero, every node can be processed at first, since it takes no time to travel from S to that specific node.

Lemma 3.3. *Let $d_{uv} = 0\ \forall u, v \in V$, $a_u > 0\ \forall u \in V$, $B > 0$ and $f^H(t) \equiv f^C(t) := f(t)$. Then the optimal solution value*

$$OPT \geq \max\left\{\sum_u a_u,\ 2a_{u^*} - f^{-1}(B)\right\}$$

time units.

Proof. It can clearly be seen that the optimal solution has to be bigger than $\sum_u a_u$, since all nodes have to be processed for the required amount of time units. Therefore, $OPT \geq \sum_u a_u$.

On the other hand, suppose that $n = 1$, such that the graph consists of one node u_1 with corresponding drilling time a_{u_1}. By Lemma 3.1, the (optimal) time to process this node equals $OPT = \max\{a_{u_1}, 2a_{u_1} - f^{-1}(B)\} \geq 2a_{u_1} - f^{-1}(B)$. If a second node u_2 is added, it can be concluded that the optimal time to process both nodes is greater or equal to $2a_{u_1} - f^{-1}(B)$, because either waiting time (if any) used to process u_1 can be used to process node u_2, or extra time is needed to process u_2. In general, when adding more nodes, the optimal maximum completion time $OPT \geq 2a_{u_1} - f^{-1}(B)$ holds. Because node u_1 can be chosen randomly, the statement also holds for $u_1 = u^*$. As a result, the optimal time to process all nodes in a graph equals $OPT \geq 2a_{u^*} - f^{-1}(B)$.

Consequently, $OPT \geq \sum_u a_u$ and $OPT \geq 2a_{u^*} - f^{-1}(B)$, which together proves the lemma that $OPT \geq \max\{\sum_u a_u,\ 2a_{u^*} - f^{-1}(B)\}$. □

Corollary. *If $f^H(t) = t$ and $f^C(t) = \alpha t$, a similar proof can be given by making use of Lemma 3.2. Hence, let $d_{uv} = 0\ \forall u, v \in V$, $a_u > 0\ \forall u \in V$ and $B > 0$. Moreover, let $f^H(t) = t$ and $f^C(t) = \alpha t$ $(0 < \alpha < 1)$. Then the optimal solution value*

$$OPT \geq \max\left\{\sum_u a_u,\ a_u + \frac{a_u - B}{\alpha}\right\}$$

time units.

As a result, for both pairs of temperature functions, lower bounds on the optimal maximum completion time are given. It is not yet known whether these lower bounds can be achieved. The following discusses when these lower bounds can be met, considering different temperature functions.

3.2.1 Identical temperature functions

In the first case, the temperature functions are assumed to be the same, meaning that $f^H(t) \equiv f^C(t) := f(t)$. One might suspect that the optimal way to process all nodes with these temperature functions, is just to start processing at a starting node for the maximum allowed of time, and consequently process the node whose temperature is the lowest and process that for the maximum amount of time. In the end, if there is any time left that has to be processed at a certain node, this can be done by waiting. This procedure will be the base of the following two theorems, where the magnitude of the node with the largest drilling time determines whether waiting times need to occur.

Theorem 3.4. *Let $d_{uv} = 0\ \forall u, v \in V$, $a_u > 0\ \forall u \in V$, $B > 0$ and $f^H(t) \equiv f^C(t) := f(t)$. Then $OPT = \sum_u a_u$ if and only if $a_{u^*} - f^{-1}(B) \leq \sum_{u \neq u*} a_u$.*

Proof. To prove this theorem, both directions have to be shown:

1. First, it is proven that if $a_{u^*} - f^{-1}(B) \leq \sum_{u \neq u*} a_u$, then OPT $= \sum_u a_u$. Since $a_{u^*} - f^{-1}(B) \leq \sum_{u \neq u*} a_u$, it can be seen that when a_{u^*} is added at both sides, $2a_{u^*} - f^{-1}(B) \leq \sum_u a_u$. Therefore, making use of Lemma 3.3, it can be seen that $OPT \geq \sum_u a_u$. In the upcoming, a procedure is given that always results in a maximum completion time of exactly $\sum_u a_u$.

   Note that if $f(a_{u^*}) \leq B$, then $f(a_u) \leq B\ \forall u \in V$, and an optimal route can easily be found by processing all nodes $u \in V$ consecutively for a_u time units, resulting in a maximum completion

time of $\sum_u a_u$ trivially. Hence, let $f(a_{u^*}) > B$. Then, node u^* can be processed for $f^{-1}(B)$ time units, resulting in its temperature after this step being $f(f^{-1}(B)) = B$. Let u_t denote the node processed in time step t. For example, in the first step, $u_1 = u^*$, since u^* was processed first. In every step from now on, choose u_t such that $u_t = \max\{u \in V, u \neq u_{t-1} : l_{ut}\}$, such that u is the node, apart from the one lastly processed, with the maximum processing time left. In every step, node u_t is processed for $f^{-1}(1)$ time units and afterwards, node u_{t+1} is processed for $f^{-1}(1)$ time units, etc. This is allowed since $f(f^{-1}(1)) = 1 \leq B$ (remember that B is a positive integer), and thus the temperature of none of the nodes exceeds B. Consequently, there is no problem with a node being too hot to process. As a result, all nodes are processed without having any waiting time, such that the maximum completion time of this procedure is exactly $\sum_u a_u$. The only thing that needs to be shown, is that there always exists some node to go to, until all nodes have been processed. Suppose $u_t = 0$ and $l_{vt} = \phi > 0$ for some $v \in V$. Then $v = u_{t-1}$ and $l_{v,t-1} = \phi + f^{-1}(1)$. As a result, $l_{u_{t-2}t-2} = f^{-1}(1)$ and $l_{u,t-2} = 0$ for all $u \neq u_{t-2}$. As a consequence, $u_{t-3} = v$ and $u_{t-4} \neq v$ etc. It can be concluded that v is alternately processed with another node. This results in the following: either $v = u_1$ or $v = u_2$. If $v = u_1$, this means that $v = u^*$. It can now be seen that

$$a_{u^*} = f^{-1}(B) + \sum_{u \neq v} a_u + \phi \geq a_{u^*} + \phi > a_{u^*},$$

which leads to a contradiction. If $v = u_2$, then it can be seen that if all nodes are alternately processed with v, there are still ϕ time units left to be processed at v. This means that

$$a_v = \sum_{u \neq v} a_u + \phi = \sum_{\substack{u \neq v \\ u \neq u^*}} a_u + a_{u^*} + \phi > a_{u^*},$$

which leads to a contradiction as well. As a result, this procedure processes all nodes with a maximum completion time of $\sum_u a_u$. Hence, since $OPT \geq \sum_u a_u$, it is derived that $OPT = \sum_u a_u$.

2. Next, it is proven that if $OPT = \sum_u a_u$, then $a_{u^*} - f^{-1}(B) \leq \sum_{u \neq u*} a_u$. This can easily be done by again making use of Lemma 3.3. By Lemma 3.3, it must be that $2a_{u^*} - f^{-1}(B) \leq \sum_u a_u$, since $\sum_u a_u$ is the optimal maximum completion time. Subtracting a factor a_{u^*} from both sides, it can directly be seen that $a_{u^*} - f^{-1}(B) \leq \sum_{u \neq u*} a_u$.

□

As a conclusion, it can be seen that it depends on the size of the node with the greatest amount of drilling time whether all nodes can be processed without any waiting time. Particularly, if the node with the greatest amount of drilling time is substantially big compared to all others combined, it is not hard to see that waiting times have to occur somehow, and consequently, an optimal maximum completion time of exactly $\sum_u a_u$ time units cannot be achieved. The following theorem states when this is the case, and what the optimal maximum completion time then exactly is.

Theorem 3.5. *Assume that* $d_{uv} = 0\ \forall u, v \in V$, $a_u > 0\ \forall u \in V$, $B > 0$ *and* $f^H(t) \equiv f^C(t) := f(t)$. *Then* $OPT = 2a_{u^*} - f^{-1}(B)$ *if and only if* $a_{u^*} - f^{-1}(B) \geq \sum_{u \neq u*} a_u$.

Proof. To prove this theorem, both directions have to be shown:

1. First, it is proven that if $a_{u^*} - f^{-1}(B) \geq \sum_{u \neq u*} a_u$, the optimal solution equals OPT $= 2a_{u^*} - f^{-1}(B)$. Similar to the proof of Theorem 3.4, when adding a_{u^*} at both sides of the inequality and using Lemma 3.3, it can be seen that $OPT \geq 2a_{u^*} - f^{-1}(B)$. By construction, the optimal route can be found as follows:

   Since $a_{u^*} - f^{-1}(B) \geq \sum_{u \neq u*} a_u$, it can be derived that $f(a_{u^*}) > B$ since $a_u > 0\ \forall u \in V$ and thus $a_{u^*} > f^{-1}(B)$. Hence, it is allowed to start by processing node u^* for $f^{-1}(B)$ time units. Now for each $v \in V \backslash \{u^*\}$, the following can be done:

   (a) If $f(a_v) \leq B$, node v can be processed for exactly a_v time units. Afterwards, node u^* has cooled down to $B - f(d_{u^*v} - a_v) = B - f(a_v) \geq 0$, which means u^* can now be processed for a_v time units, such that the temperature at node u^* becomes $B - f(a_v) + f(a_v) = B$ again.

   (b) If $f(a_v) > B$, then v has to be processed more than once. At first, it can be processed for $f^{-1}(B)$ time units, resulting in the temperature of v becoming $f(f^{-1}(B)) = B$. Then, node u^* has cooled down to $B - f(d_{vu^*} + f^{-1}(B)) = B - B = 0$, which means that u^* can now be processed for $f^{-1}(B)$ time units again. Afterwards, node v has cooled down to $B - f(d_{u^*v} - f^{-1}(B)) = B - B = 0$ as well. This procedure can be repeated until $f(a_v) \leq B$. Then, the previous method can be applied where $f(a_v) \leq B$ to get node v processed completely.

   As a result, all nodes can be processed in this way. This means that the total time spent is $f^{-1}(B) + 2\sum_{u \neq u^*} a_u$. Moreover, one can derive that node u^* has being processed for $f^{-1}(B) + \sum_{u \neq u^*} a_u$ time units at total, and its temperature at the end of the procedure is exactly B. Keep in mind that no waiting times occurred at any node yet. The time left to process node u^* is $a_{u^*} - f^{-1}(B) - \sum_{u \neq u^*} a_u \geq 0$ by assumption. This means that this amount of processing time, as well as waiting time, is needed to get node u^* enough processed in the end. Hence, the additional time that has to be spent at the last node is $2(a_{u^*} - f^{-1}(B) - \sum_{u \neq u^*} a_u)$. All together gives that the total time spent is

$$f^{-1}(B) + 2\sum_{u \neq u^*} a_u + 2\left(a_{u^*} - f^{-1}(B) - \sum_{u \neq u^*} a_u\right) =$$
$$f^{-1}(B) + 2\sum_{u \neq u^*} a_u + 2a_{u^*} - 2f^{-1}(B) - 2\sum_{u \neq u^*} a_u =$$
$$2a_{u^*} - f^{-1}(B).$$

   As a result, a procedure is shown which has a maximum completion time of $2a_{u^*} - f^{-1}(B)$. Hence, it is derived that $OPT = 2a_{u^*} - f^{-1}(B)$.

2. Second, it is proven that if the optimal solution equals $OPT = 2a_{u^*} - f^{-1}(B)$, then $a_{u^*} - f^{-1}(B) \geq \sum_{u \neq u*} a_u$. Again, this can easily be done by again making use of Lemma 3.3. By Lemma 3.3, it must be that $2a_{u^*} - f^{-1}(B) \geq \sum_u a_u$, since $2a_{u^*} - f^{-1}(B)$ is the optimal maximum completion time. Subtracting a factor a_{u^*} from both sides, the desired result can directly be seen that $a_{u^*} - f^{-1}(B) \geq \sum_{u \neq u*} a_u$.

□

Corollary. *If* $a_{u^*} - f^{-1}(B) = \sum_{u \neq u*} a_u$, *then*

$$\begin{aligned} OPT &= 2a_{u^*} - f^{-1}(B) \\ &= a_{u^*} + a_{u^*} - f^{-1}(B) \\ &= a_{u^*} + \sum_{u \neq u*} a_u \\ &= \sum_u a_u. \end{aligned}$$

As a conclusion, for the case that the temperature functions are the same, characterizations are given whether all nodes can be processed in $\sum_u a_u$ time units, so without any waiting time, or not. If this cannot be done, it is shown that $2a_{u^*} - f^{-1}(B)$ time units are needed to process all nodes.
In the next step, the same is done to know what happens when cooling down happens slower than heating up. It has already been mentioned that in this case, it is assumed that $f^H(t) = t$ and $f^C(t) = \alpha t$ $(0 < \alpha < 1)$

3.2.2 Linear temperature functions with slow cooling down

When nodes cool down slower than they heat up, the analysis of the previous does not hold, since both functions are not the same. When a node has been processed, it needs more time to cool down. As a result, most probably other conditions need to be met to know whether all nodes can be processed in $\sum_u a_u$ time units or not.

By looking at the previous section, and compare this to the corollary of Lemma 3.3, one might have the following conjecture:

Conjecture. *Assume that* $d_{uv} = 0\ \forall u, v \in V$, $a_u > 0\ \forall u \in V$, $B > 0$ *and* $f^H(t) = t$ *and* $f^C(t) = \alpha t$, $0 < \alpha < 1$. *Then* $\frac{a_{u^*} - B}{\alpha} \leq \sum_{u \neq u*} a_u$ *if and only if* $OPT = \sum_u a_u$. *On the other hand,* $\frac{a_{u^*} - B}{\alpha} \geq \sum_{u \neq u*} a_u$ *if and only if* $OPT = a_{u^*} + \frac{a_{u^*} - B}{\alpha}$.

One might again suspect that the optimal way to process all nodes with these temperature functions, is the same as the procedure given when the temperature functions are identical: start processing at a starting node for the maximum allowed of time, and consequently process the node whose temperature is the lowest and process that for the maximum amount of time. In the end, if some nodes have still not been fully processed, this is solved by waiting and processing consecutively. The following example shows that this procedure does not always result in the optimal maximum completion time regarding the conjecture.

Example 3.1. *Consider following graph.*

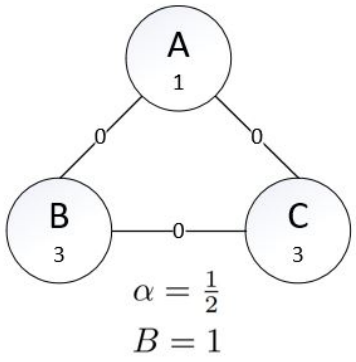


Figure 3: An example graph with 3 nodes, where $B = 1$ *is seen as the maximum allowed temperature of a node and* $\alpha = \frac{1}{2}$ *is being used as the cooling down factor.*

With the given procedure, there is a choice between two starting nodes: either starting at A or starting at B. Note that starting at C is equivalent to starting at node B.

When starting at A and using the given procedure, the following happens:

A	1							
B		1		α		α^3		...
C			1		α^2		α^4	

Table 1: The procedure applied to the graph when starting at node A.

Nodes B and C can in this case be interchanged in order. Node B is now being processed for

$$1+\alpha+\alpha^3+\alpha^5+\ldots = 1+\sum_{i=1}^{\infty}\alpha^{2i-1} = 1+\frac{1}{\alpha}\sum_{i=1}^{\infty}\left(\alpha^2\right)^i = 1+\frac{1}{\alpha}\frac{\alpha^2}{1-\alpha^2} = 1+\frac{\alpha}{1-\alpha^2},$$

and node C being processed for

$$1+\alpha^2+\alpha^4+\ldots = \sum_{i=0}^{\infty}\alpha^{2i} = \sum_{i=0}^{\infty}\left(\alpha^2\right)^i = \frac{1}{1-\alpha^2}.$$

Both series converge since $\alpha^2 < 1$. Because α in this example is equal to $\frac{1}{2}$, node B has been processed for $\frac{5}{3}$ time units and node C has been processed for $\frac{4}{3}$ time units, and the temperature of both nodes at the end equals 1. Hence, since waiting times have to occur somehow, the maximum completion time is greater than $\sum_u a_u$, which is 7 time units.

On the other hand, if node B (or C) is the starting node, the following two orders can apply, depending on the choice in the second step:

A			1					
B	1			1		α		...
C		1			1		α^2	

A		1						
B	1			1		α^2		...
C			1		α		α^3	

Table 2: The procedure applied to the graph when starting at node B.

The first order processes node B and C for $\frac{8}{3}$ and $\frac{7}{3}$ time units respectively, whereas the second order processes node B and C for $\frac{7}{3}$ and $\frac{5}{3}$ time units respectively. Both are better than the order given in Table 1, but there is still waiting time needed to get all nodes processed. Therefore, this procedure leads to a maximum completion time of greater than 7 time units as well.

Applying the conjecture to Example 3.1, it can be checked that

$$\frac{a_{u^*}-B}{\alpha} = \frac{3-1}{\frac{1}{2}} = 4 = 3+1 = \sum_{u\neq u*} a_u,$$

and thus the optimal maximum completion time must be equal to $\sum_u a_u = a_{u^*} + \frac{a_{u^*}-B}{\alpha} = 7$ time units. Therefore, the question arises whether this conjecture is true in the first place. The procedure mentioned earlier might not be the way to achieve this optimal maximum completion time, and it might be hard to see that all nodes actually *can* be processed without any waiting times. Nevertheless, it can be done in 7 time units making use of small fractions and infinitely many processing steps in the following way:

A											1									
B	...		α^6		α^4		α^2		1			1		α		α^3		α^5		α^7	
C		α^7		α^5		α^3		α		1			1		α^2		α^4		α^6		...

One can check that in this way, the temperature of a node does not exceed the maximum of $B = 1$, and all nodes are processed for the required drilling time. For example, looking at node A, it can be seen that it is processed for

$$\sum_{i=1}^{\infty} \alpha^{2i-1} + 1 + 1 + \sum_{i=1}^{\infty} \alpha^{2i} = \frac{1}{\alpha} \sum_{i=1}^{\infty} \left(\alpha^2\right)^i + 2 + \sum_{i=1}^{\infty} \left(\alpha^2\right)^i = \frac{1}{\alpha} \frac{\alpha^2}{1-\alpha^2} + 2 + \frac{\alpha^2}{1-\alpha^2}$$
$$= 2 + \frac{\alpha + \alpha^2}{1-\alpha^2} = 2 + \frac{\alpha(\alpha+1)}{(1-\alpha)(\alpha+1)} = 2 + \frac{\alpha}{1-\alpha}$$

Noticing that $\alpha = \frac{1}{2}$ in this example, this expression solves to

$$2 + \frac{\alpha}{1-\alpha} = 2 + \frac{\frac{1}{2}}{1-\frac{1}{2}} = 2 + 1 = 3,$$

which is exactly the drilling time needed to process node A. Node B can be checked in a similar way, and it is trivial that node C is fully processed. As a result, all nodes are processed for the required drilling times and the maximum of B is not exceeded. Hence, this is a proper way to process all nodes without any waiting time.

As a result, the conjecture might be a true statement, which challenges us to prove it. Unfortunately, the proof cannot be constructed similar than the proofs of Theorems 3.4 and 3.5, since it is not sure whether nodes have been cooled down at a certain moment. As can be seen from the example, infinitely many processing steps and small fraction may be key to proof the conjecture. In section 5, this is discussed to a greater extent.

3.3 Big distances

Another special case of the ITSP, is when distances are substantially large compared to the drilling times. If the distances are very large, many travels will result in a large maximum completion time. Therefore, one can imagine that the best option would be to visit all nodes exactly once, and processing each node in this one visit. The lemmas stated in section 3.1 tell us what time it takes to processed a node in one visit. Since we have to visit every node once, a TSP tour is needed to find the optimal route. In this section, we assume the temperature functions to be linear, meaning $f^H(t) = f^C(t) = t$.

At first, the calculations are simplified by assuming that $d_{uv} = \gamma > 0 \ \forall u, v \in V$, which means that all distances between nodes are the same. This creates insight to prove a theorem for cases where distances are actually *not* the same. To make things interesting, let $a_{u^*} > B$, since otherwise, it can easily be seen that a TSP tour is the optimal way (which actually is just a tour passing all nodes, since all distances are the same). Also note that the starting node does not have a drilling time itself.

Theorem 3.6. *Let $d_{uv} = \gamma > 0 \ \forall u, v \in V$, $0 < B < a_{u^*}$. If $B \leq \gamma$, then the optimal maximum completion time equals*

$$OPT = \gamma(n+1) + \sum_{u} \max\{a_u, 2a_u - B\}$$

time units.

Proof. Let the following way be proposed as a solution: start at a starting node s and visit all nodes exactly once, where each node is processed (with some waiting times if necessary) until the required drilling time is met. The total traveling time then equals $\gamma(n+1)$ and the total drilling time equals $\max\{a_u, 2a_u - B\}$ for every node u by Lemma 3.1. Hence, the maximum completion time equals $\gamma(n+1) + \sum_u \max\{a_u, 2a_u - B\}$. For a contradiction, suppose that δ extra travels decreases the maximum completion time. These extra travels increase the maximum completion time with $\delta\gamma$ of traveling times, and decreases the waiting time of a certain node u with at most B. As a result, the maximum completion

time T^* is now

$$\begin{aligned} T^* &\geq \gamma(n+1) + \sum_u \max\{a_u, 2a_u - B\} + \delta\gamma - \delta B \\ &\geq OPT + \delta(\gamma - B) \\ &\geq OPT \end{aligned}$$

□

Now it is desired to state a theorem that does *not* use $d_{vu} = \gamma > 0 \ \forall v, u \in V$. Instead, it is now assumed that $d_{uv} > 0 \ \forall v, u \in V$, so it need not to be constant anymore. By intuition, one could derive that when the distances become 'big', lots of travels do not result in a small completion time. Instead, a TSP tour with waiting at all nodes is optimal, as already mentioned. The question that arises is: how big need those distances to be such that this holds? The following gives an answer to this question.

Theorem 3.7. *Let $d_{uv} > 0 \ \forall u, v \in V$, $a_u > 0 \ \forall u \in V$ and $B > 0$. If for all $a_u > B$ it holds that $d_{uv} + d_{uw} \geq d_{vw} + \min\{a_u - B, B\} \ \forall v, w \in V \backslash \{u\}$, then a TSP tour with waiting at each node is the optimal solution.*

Proof. Lets consider the TSP solution where each node is processed in exactly one processing step. Suppose that this TSP tour including the waiting and processing times takes T^* time units. Note that for all nodes $u \in V$ for which $a_u \leq B$, adding an extra visit to u would not be useful, since u can already be fully processed in one processing step. This could be done faster if visiting a node u for which $a_u > B$ twice results in a smaller maximum completion time. This is done as follows: suppose that the edge $(v, w),\ v \neq u,\ w \neq u$ is in the TSP tour. Delete this edge (v, w) and replace it by the edges (u, v) and (u, w) to obtain a route where all nodes are visited once, and u is now visited twice. Since u is visited twice, waiting times at u (since $a_u > B$ and a_u was processed in one visit, there were waiting times at u) can be omitted. If $B < a_u \leq 2B$, $a_u - B$ time units of waiting time can be omitted. If $a_u > 2B$, at most B time units of waiting time can be omitted. As a result, exactly $\min\{a_u - B, B\}$ time units of waiting time can be omitted. Compared to T^*, the time needed to travel from v to w and $\min\{a_u - B, B\}$ time units of waiting time are gone. On the other hand, the time needed to travel from u to v and from u to w have to be added. As a result, the new maximum completion time T equals

$$\begin{aligned} T &= T^* + d_{uv} + d_{uw} - d_{vw} - \min\{a_u - B, B\} \\ &\geq T^* \end{aligned}$$

As a result, an extra visit to u does not result in a smaller maximum completion time. Hence, a TSP tour with waiting at each node is the optimal solution. □

4 Models

In this section, two models regarding the ITSP are being analysed. One model deals with positive distances, where the temperature functions are $f^H(t) = t$ and $f^C(t) = \alpha t,\ 0 < \alpha < 1$. Because the ITSP is NP hard, this model works only for small instances. The second model gives a solution to the ITSP where distances are equal to zero. This solution might not always be the optimal one, which is explained later on.

4.1 Model with positive distances

In the first model, it is assumed that there is a fixed number of processing steps s_{max}. As a result, $A = \{1, 2, ..., s_{max}\}$ are the different processing steps that are taken in this problem. The definition of a processing step is given in section 2.2. In this model, the set of nodes V is used as a set of natural numbers, meaning $V = \{0, 1, ..., n\}$. In addition, it is assumed that a_v, d_{vu} and B are all integer valued, and both heating up and cooling down are linear. In particular, it is used that $f^H(t) = t$ and $f^C(t) = \alpha t$, where $0 < \alpha < 1$. Note that if $\alpha = 1$, both heating up and cooling down are equal.

Before the model as a whole is given, all variables that are used are outlined. Six variables are used in this model:

- x_{us}: a binary variable which is 1 if node $u \in V$ is processed at processing step $s \in A$ and 0 otherwise.
- y_{uvs}: a binary variable which is 1 if there is a travel from node u to node v between processing step s and $s + 1$, and 0 otherwise.
- p_{us}: a nonnegative variable which corresponds to the processing time of node $u \in V$ in processing step $s \in A$.
- b_s: a nonnegative variable which indicates the time at the beginning of processing step $s \in A$.
- z_{us}: a nonnegative variable in the range $[0, B]$ which represents the temperature of a node $u \in V$ at the *beginning* of processing step $s \in A$.
- w_{us}: a nonnegative variable which represents the waiting time of node $u \in V$ at processing step $s \in A$.

Now that the variables are known, the model can be shown:

$$\text{minimize} \quad b_{s_{max}} + \sum_{u \in V} p_{u,s_{max}} + \sum_{u \in V} w_{u,s_{max}}$$

subject to

$$\sum_{u \in V} x_{us} \leq 1 \qquad \forall s \in A \tag{1}$$

$$x_{ut} = \sum_{v \in V} y_{uvs} \qquad \forall u \in V, s \in A \mid s < s_{max} \tag{2}$$

$$x_{us} = \sum_{v \in V} y_{vu,s-1} \qquad \forall u \in V, s \in A \mid s > 1 \tag{3}$$

$$\sum_{s \in A} p_{us} = a_u \qquad \forall u \in V \tag{4}$$

$$x_{1,1} = x_{1,s_{max}} = 1 \tag{5}$$

$$p_{us} \leq x_{us} B + \alpha w_{us} \qquad \forall u \in V, s \in A \tag{6}$$

$$w_{us} \leq x_{us} \frac{a_{u^*}}{\alpha} \qquad \forall u \in V, s \in A \tag{7}$$

$$z_{us} + p_{us} \leq B + \alpha w_{us} \qquad \forall u \in V, s \in A \tag{8}$$

$$b_{s+1} \geq b_s + \sum_{u \in V} p_{us} + \sum_{u,v \in V} d_{uv} y_{uvs} + \sum_{u \in V} w_{us} \qquad \forall s \in A \mid s < s_{max} \tag{9}$$

$$z_{u,s+1} \geq z_{us} + p_{us} - \alpha(b_{s+1} - (b_s + p_{us})) \qquad \forall u \in V, s \in A \mid s < s_{max} \tag{10}$$

$$x_{us} \in \{0,1\} \qquad \forall u \in V, s \in A \tag{11}$$

$$y_{uvs} \in \{0,1\} \qquad \forall u, v \in V, s \in A \tag{12}$$

$$p_{us} \geq 0 \qquad \forall u \in V, s \in A \tag{13}$$

$$s_s \geq 0 \qquad \forall s \in A \tag{14}$$

$$z_{us} \in [0, B] \qquad \forall u \in V, s \in A \tag{15}$$

$$w_{us} \geq 0 \qquad \forall u \in V, s \in A \tag{16}$$

At first, the object function is explained. The goal of the ITSP is to minimize the total time that is needed to process all nodes. The variable b_s is used in the objective function, since it keeps track of the time spent before arriving at a certain processing step. Hence, $b_{s_{max}}$ can be used to express what time was needed to get to the last processing step. The only thing that has to be added now, is the time spent in this last step, which can only be processing time, either with waiting times or without.

Next, all constraints stated above are outlined. Constraint (4.1) ensures that in every processing step, at most one node is processed. Constraints (4.2) and (4.3) take care of traveling between nodes. Constraint (4.2) can be explained as follows: if a node u is processed at processing step s (such that $x_{us} = 1$), then in the next step, there has to be a visit at some (other) node v. Hence, when summing y_{uvs} over all v, an outcome of exactly 1 is needed as well. Analogously, if a node u is not visited in processing step s (which means that $x_{us} = 0$), then there may *not* be a travel from u to v in the next step, meaning that when summing y_{uvs} over all v, an outcome of 0 is needed. As a conclusion, this constraint can be seen as the constraint that ensures that if a node u is visited in some processing step s, it can only have *one* node to go to in the next step. Constraint (4.3) is quite similar to constraint (4.2): if node u is processed at processing step s (such that $x_{us} = 1$), then node u is reached from only *one* node in the previous step. Hence, when summing $a_{vu,s-1}$ over all v, the outcome has to be 1 as well. A similar argument holds when $x_{us} = 0$. As a conclusion, this constraint can be seen as the constraint that ensures that if a node u is visited in some processing step s, it can only have *one* predecessor.

Of course, every node needs to be processed for the required amount of time units a_u. In this model, this

gives rise to constraint (4.4). It is also assumed that there is one starting and end node, which can be seen in constraint (4.5). In this model, node 1 is this node. If another node is desired to be the starting and end node, one can reorder the nodes easily.

If waiting times would not be included, constraint (4.6) would be just $p_{ut} \leq x_{ut}B$, such that there is no processing time at a particular node when it is not visited, and it is processed for at most B time units when it actually *is* visited. Including waiting times would preferable give rise to the constraint $p_{us} \leq x_{us}(B + \alpha w_{us})$. It can be seen that an extra factor αw_{us} is added, which can be explained as follows: if an extra amount of w_{us} time units is waited at this node, the node has cooled down with a temperature of αw_{us}, so that this amount of time units can extra be processed at this node. Unfortunately, the model would not be linear any more, since the factor $x_{us}w_{us}$ pops up, which is a product of two variables. Therefore, constraint (4.6) can be written as $p_{us} \leq x_{us}B + \alpha w_{us}$, and an extra constraint (constraint (4.7)) is added to make sure that waiting times only occur at nodes that are processed at a certain processing step. Otherwise, multiple nodes can be processed because there can be waited at a node (and thus be processed) when it is not actually being processed at that processing step, meaning $x_{us} = 0$ and $w_{us} > 0$ for some u. Therefore, w_{us} needs to be zero if x_{us} equals zero, which can be done by adding the constraint $w_{us} \leq cx_{us}$, where c is a constant that is large enough. The maximum waiting time that can occur, is when a_{u*} is processed in one go. In this case, $\alpha w_{us} \leq a_{u^*}$, such that $w_{us} \leq \frac{a_{u^*}}{\alpha}$. Together gives constraints (4.6) and (4.7).

Constraint (4.8) is the temperature constraint. In general, the temperature that a node has at the beginning of a certain processing step plus the processing time of this node in this processing step may never exceed B at all time, which leads to $z_{us} + p_{us} \leq B$. Again, when waiting times are included, an extra factor αw_{us} is added on the right-hand side, for the same reason as in constraint (4.6). Constraint (4.9) defines the arrival times. Since $b_{s_{max}}$ is part of the object function, minimal values are used in the inequality of this constraint. The constraint states that the arrival time in processing step $s+1$ is equal to the time at the beginning of the previous processing step, together with what happens in processing step s. What can happen here, is processing, traveling and/or waiting. All together gives constraint (4.9).

The most complicated constraint of the model is constraint (4.10). This constraint defines temperature of all nodes at all processing steps and can be explained as follows: the temperature of a node in processing step $s+1$ depends on the temperature of this node in processing step s and whatever happens in between. Increasing of this temperature can only happen when this node is processed during processing step s. Decreasing of temperature can be expressed as the difference of the time at the beginning of processing step $s + 1$ and the beginning of processing step s. Note that when a node is processed between $s + 1$ and s, this processing time needs to be extracted from this difference. Hence, for a node u at time $s + 1$, its temperature can increase by p_{us} and decrease by $\alpha(b_{s+1} - (b_s + p_{us}))$. The result can be seen in constraint (4.10). Constraints (4.11) - (4.16) are range constraints for the variables. All constraints, together with the object function, show the model that finds the optimal solution to the ITSP with fixed processing steps. A similar model, one without explicit waiting times and linear cooling down and heating up (meaning $\alpha = 1$) has already been studied [8].

The disadvantage this model has, is that it assumes a fixed amount of processing steps. Beforehand, it is not known how many of these processing steps the optimal solution has. Therefore, an algorithm is implemented to find this as quick as possible. Before this algorithm can be stated, an interesting property of this model is explained.

Let $C(s_{max}) = \sum_{u} \sum_{s=2}^{s_{max}} \delta_{us}$ be the amount of consecutive drills in the model, where

$$\delta_{us} = \begin{cases} 1 & \text{if } p_{us}p_{u,s-1} = 1, \\ 0 & \text{otherwise.} \end{cases}$$

Note that $p_{us}p_{u,s-1} = 1$ if and only if $p_{us} = 1$ and $p_{u,s-1} = 1$, which means that the same node is being processed consecutively, which is called a *consecutive drill*. Now, the following lemma can be stated.

Lemma 4.1. *$\overline{s}$ is the optimal amount of processing steps if and only if $C(\overline{s}) = 0$ and for all $m \in \mathbb{N} \geq 1$, $C(\overline{s} + m) = m$.*

This lemma will not be proven, but an easy explanation can be given: when a fixed amount s of processing steps, two things can occur: either there are no consecutive drills, or there are $m \geq 1$ consecutive drills. In the first case, it cannot be known whether s is the optimal amount of processing steps. It may be the case that $s + n$ processing steps, with $n \geq 1$, results in a lower maximum completion time since more processing steps can reduce the waiting time.

By Lemma 4.1, all solutions with $s + m$ processing steps have to be checked whether there are m consecutive drills. Of course, this is undesirable. Therefore, the algorithm increases the amount of processing steps by 2 in each step, and checks whether consecutive drills occur. The full algorithm can now be shown.

Algorithm 1 Optimal Amount of Processing Steps

```
1:  s_max ← n + 1
2:  solve ITSP
3:  while C(s_max) = 0 do
4:      s_max ← s_max + 2
5:      solve ITSP
6:      if C(s_max) = 2 then
7:          break
8:      else if C(s_max) = 1 then
9:          s_max ← s_max − 1
10:         solve ITSP
11:         break
12: if C(s_max) ≠ 0 then
13:     s_max ← s_max − C(s_max)
14:     solve ITSP
```

It can be seen that it uses only the amount of consecutive drills to determine whether the solution is optimal or not, instead of the maximum completion time itself. The utility of this algorithm is discussed in section 5. To finish this section, lets see what this model including the algorithm exactly does. This is done by implementing the model in AIMMS. The complete code can be found in the Appendix. Consider the following graph:

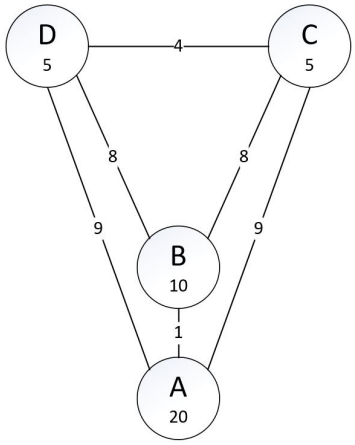
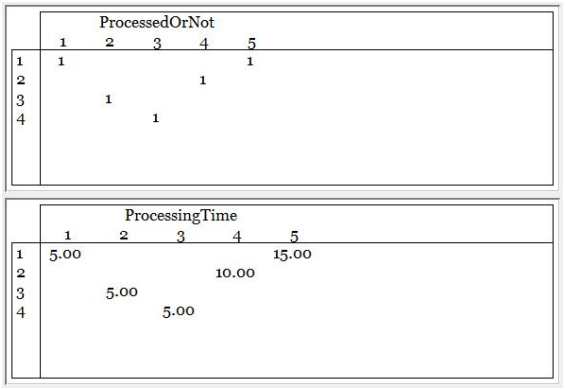


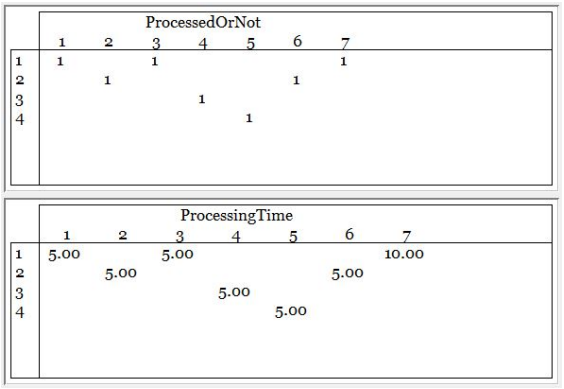
Figure 4: Example graph with 4 nodes. The value inside the node corresponds to the drilling time of that node.

Note that for the model, we use the value 1 for node A, 2 for node B, etc. Figure 5 shows the whole process to find the optimal solution. *ProcessedOrNot(u,s)* is used as x_{us} and *ProcessingTime(u,s)* is used as p_{us}. The model, together with the algorithm, starts by finding the optimal solution with 5

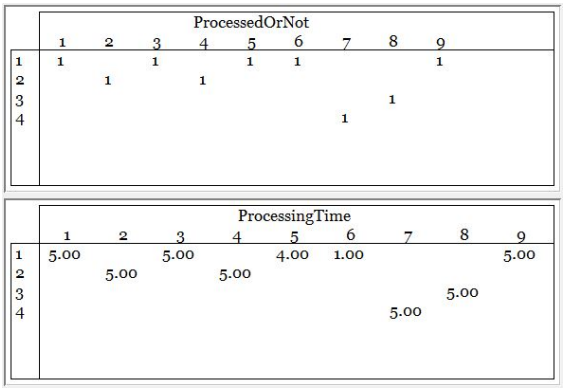
processing steps, starting and ending at node A. In fact, this corresponds to a TSP-tour with processing (and waiting if necessary) at each node in one go. The result can be seen in Figure 5a. Since no consecutive drills occur, the amount of processing steps is increased by 2 and the solution is calculated. The result is shown in 5b, the maximum completion time is lower, and still no consecutive drills occur. Hence, the amount of processing steps is increased by 2 again. Now, in Figure 5c, it can be seen that there is one consecutive drill, namely at processing steps 5 and 6. As a result, it can be concluded that 8 processing steps results in the optimal maximum completion time. This result is shown in Figure 5d, and it can be checked that the optimal maximum completion time equals 66 time units.



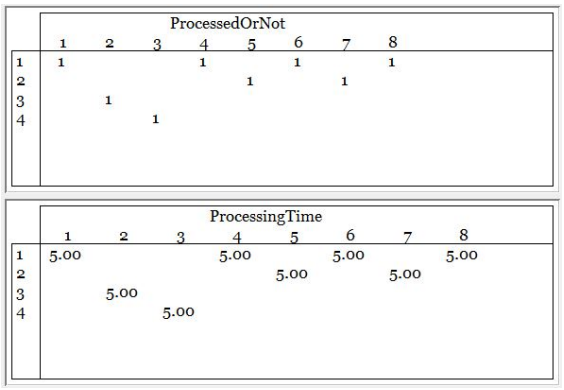
(a) *The optimal solution with 5 processing steps, taking 77 time units in total.*



(b) *The optimal solution with 7 processing steps, taking 69 time units in total.*



(c) *The optimal solution with 9 processing steps, taking 66 time units in total, and having one consecutive drill.*



(d) *The optimal solution with 8 processing steps, taking 66 time units in total. This is the optimal solution to the graph in figure 4.*

Figure 5: Visualisation of Algorithm 1.

4.2 Model with no distances

Secondly, a less complex model can be set up to give a solution to the ITSP where it is assumed that distances are equal to zero. This model transforms the input, which is done in two ways. This model does not find the optimal solution surely, since it adapts the input. Nevertheless, when the amount of nodes increases, both models find in most cases the optimal solution, as can be seen later on.

As being said, before the model can actually be used, the drilling times are transformed. This is done in two different ways:

Method 1: let a_u represent the drilling time of node u. Then unique integers q and r can be found such that $a_u = qB + r$ with $0 \leq r < B$. This means that a_u can be divided in q pieces of B and one piece of r. These pieces are referred to as *jobs*. As a conclusion, a_u can be divided into $q+1$ parts where $u_1 = u_2 = ... = u_q = B$ and $u_{q+1} = r$. For example, if $a_u = 26$ for some $u \in V$ and $B = 5$, then $u_i = 5$ for $i = 1, 2, ..., 5$ and $u_6 = 1$.

Method 2: The drilling times can also be divided in parts which are approximately the same. It depends on B in how many jobs a_u has to be split, since each part may not exceed B. Therefore, let $\lceil \frac{a_u}{B} \rceil = \lambda_u$. Now, the drilling time a_u is divided into λ_u jobs. To get those parts as equally sized as possible, let for every node $u \in V$, $u_1 = u_2 = ... = u_{\lambda_u - 1} = \lceil \frac{a_u}{\lambda_u} \rceil$ and $u_{\lambda_u} = a_u - (\lambda_u - 1)\lceil \frac{a_u}{\lambda_u} \rceil$. For example, if again $a_u = 26$ for some $u \in V$ and $B = 5$, one can check that $\lambda_u = 6$ and thus for u, it holds that $u_i = 5,\ i = 1, 2, ..., 5$ and $u_6 = 1$.

Moreover, the set of all jobs is denoted by J and the set of all time steps is denoted by T. Two variables are used in this model:

- $x_{u_i t}$: a binary variable which is 1 if job $u_i \in J$ is started at time $t \in T$, and 0 otherwise.
- w: a free variable that is used to determine the maximum completion time of the model.

With these two variables, the following model can be set up:

$$\text{minimize} \quad w$$

subject to

$$\sum_{t \in T} x_{u_i t} = 1 \qquad \forall u_i \in J \tag{17}$$

$$\sum_{u_i \in J} \sum_{s=t-u_i}^{t-1} x_{u_i s} \leq 1 \qquad \forall t \in T \tag{18}$$

$$x_{u_i t} \leq \sum_{s=0}^{t-u_{i-1}-\lceil \frac{u_{i-1}}{\alpha} \rceil} x_{u_{i-1} s} \qquad \forall u_i \in J, t \in T \mid t > 1 \tag{19}$$

$$w \geq t x_{u_i t} + u_i \qquad \forall u_i \in J, t \in T \tag{20}$$

$$x_{u_i t} \in \{0, 1\} \qquad \forall u_i \in J, t \in T \tag{21}$$

Constraint (4.17) ensures that all jobs are being processed. All jobs have to appear somewhere in the solution, because all nodes have to be processed long enough, which means that all jobs have to be processed in the end. Constraint (4.18) ensures that at most one job is being processed at a certain time step. Let t be fixed, then all jobs u_i are checked whether somewhere in the interval $[t - u_i, t - 1]$ this job is done. This may only appear at one job at the same time, which automatically leads to constraint (4.18).

Constraint (4.19) ensures that no job is being done to one node if the node has not fully cooled down after the previous job. Lets take a look at job i of node u. Job u_i can only be started if u is fully cooled down after job u_{i-1}. This means that it has to be checked whether job u_{i-1} has started early enough. Note that when job u_{i-1} is started at a particular time, it takes $u_{i-1} + \frac{u_{i-1}}{\alpha}$ time units to get node u fully cooled down. Because $T \subseteq \mathbb{N}$ and $\frac{u_{i-1}}{\alpha}$ might not be an integer, it can be seen that $u_{i-1} + \lceil \frac{u_{i-1}}{\alpha} \rceil$ time is needed. Hence, constraint (4.19) handles correctly with cooling down of all nodes.

In addition, constraint (4.20) keeps track of the total time used to process all jobs. $tx_{u_i t}$ denotes the starting time of job u_i and u_i represents the time needed for job u_i. Now if $w \geq tx_{u_i t} + u_i$ for all $u_i \in J$ and $t \in T$, then w is greater or equal to the total time it takes to process all jobs. Because the object function is to minimize w, the model indeed minimizes the maximum completion time. At last, constraint (4.21) is the range constraint for $x_{u_i t}$.

Note that this solution might not be optimal, because the division of the drilling times into jobs might not be the best way to solve the ITSP. Nevertheless, the solution is very close to the optimal one.

4.3 Comparing both methods of Model 4.2

Since two models have been proposed, the question arises which of the two is 'better'. Better in this case means that the outcome of the model is close to the optimal value, which is more or less known for cases where there are no distances. If $\alpha = 1$, both temperature functions are the same, and Lemma 3.3 tells us what the optimal maximum completion time exactly is. For cases where $0 < \alpha < 1$, the conjecture can be applicable. Since this is a conjecture, and it is not fully sure whether it is true, $\alpha = 1$ is used to compare this model to the known optimal maximum completion time.

Which of the models is better, depends on a lot of parameters. A different choice of the value of α may result in one of the models being significantly better. For example, it might be the case that for α very small, one of the methods is considerable faster or closer to the actual optimal value than the other. Moreover, it may be the case that the value of B, or the composition of the instance is better for one model over the other. With the latter, it is meant that maybe instances with a extremely large value for a_{u^*} compared to the others may result in one of the models being better.

In this paper, it is only investigated which of the models is better for small instances. These are instances with less than 10 nodes. The instances themselves are randomly created with the drilling times being random selected from the interval $[1, 30]$, and B being randomly selected from the interval $[4, 10]$. Moreover, $\alpha = 1$ is taken, such that Theorems 3.4 and 3.5 can be used to find out whether the optimal solution is found. By making use of these theorems, we know what the *expected optimal completion time (EOCT)* exactly is. For each amount of nodes, 100 instances are created, and for both models, the expected optimal completion time is divided by the outcome to see how close it is. Note that this fraction can never be bigger than 1, and if this fraction equals 1, the model has found a solution that is optimal. If this fraction is strictly less than 1, the model has not found the optimal solution. The results are shown in the table in Appendix A. As can be seen in Figure 6, the accuracy of both models is very high, and increases when the amount of nodes is increased. Moreover, the first model is overall slightly better than the second model for small instances.

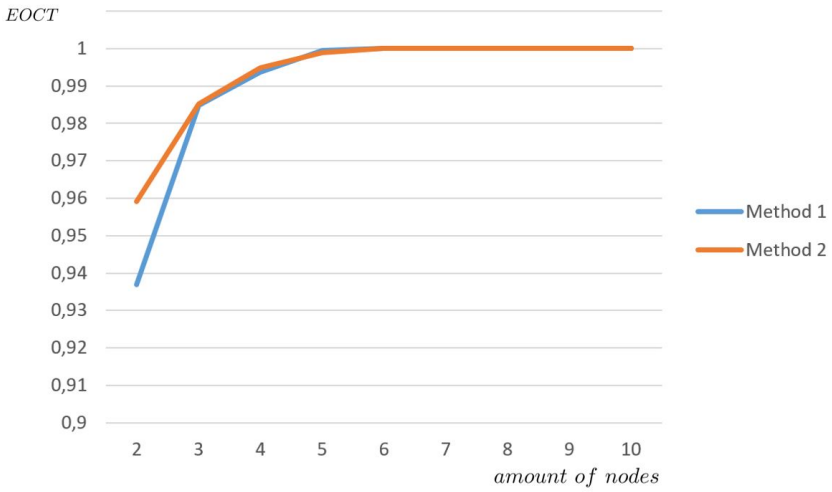


Figure 6: The outcomes of Method 1 and Method 2 compared to the expected optimal completion time (EOCT).

The figure might insinuate that these models work quite well for instances where distances are equal to zero. A discussion on these models is given in section 5. As with the previous model, the programming code can be found in Appendix B.

5 Discussion and Recommendation

In this section, the results of this paper are discussed, and some recommendations are given for further research. The first thing this paper has outlined, were cases where graphs consists of only one node. For the two pairs of temperature functions discussed in this paper, the optimal solution is given to process this one node. When graphs were extended to multiple nodes, these lemmas were used to prove (lower bounds on) the optimal maximum completion time.

The first extension that is analysed, are graphs were all distances are equal to zero. For instances where the temperature functions are identical, it is proven whether all nodes can be processed without any waiting time, or waiting time is needed to process all nodes in the optimal way. In the latter case, it is also shown that an exact optimal maximum completion time can be found. Moreover, in the proofs of these theorems, a procedure is given to find such an optimal solution. Therefore, when distances between nodes are zero, it is fully known what the optimal maximum completion time is, by only checking one simple condition.

On the other hand, when the temperature functions are linear, but cooling down happens slower, a conjecture is given. This conjecture is based on what is known about identical temperature functions. For further research, one can try to prove the conjecture stated in section 3.2.2, or maybe trying to find a counter example. The procedure to find the optimal solution to Example 3.1, with infinitely many processing steps and small fractions, might be inspiration for a proof.

In general, zero distances are interesting to be studied. In cases where distances are negligible compared to the drilling times, the theorems (and maybe the conjecture) in which distances are assumed to be zero, can be applied. In fact, these provide lower bounds on the optimal solution. Therefore, when using very small distances and identical temperature functions, the procedure used to prove both Theorems 3.4 and 3.5 work to find a feasible solution. As a remark, no temperature issues occur when there are positive distances between the nodes instead of zero distances. Therefore, instead of making use of the full model proposed in section 4.1, which is very complex for large instances, one can use the procedures of these theorems to find a feasible solution, which is close to the optimal one because of the negligibly small distances.

However, when distances are not zero, Theorem 3.7 shows what condition needs to hold such that it is known for sure that a TSP tour is optimal. Multiple approaches are known to approximate the optimal solution to the TSP for big instances [9]. Hence, this creates an efficient solving method, because Lemma 3.1 states how long it takes to process one node in one go. Together, this solves the ITSP in the cases where distances are very big.

In addition, the model in section 4.1 finds the optimal solution when considering a fixed amount of processing steps and positive distances. This model, together with Algorithm 1, finds the optimal solution to the ITSP. Improving this algorithm may lead to a lower maximum completion time. In further studies, this can be investigated to find the optimal solution faster.

The model in section 4.2 finds an solution to the ITSP where distances are assumed to be zero, by making use of two different methods to split the drilling times into separate jobs. It can be seen that these solutions are almost optimal when the amount of nodes increase in Figure 6. As a remark, both methods can be compared otherwise. For example, assuming the conjecture to be true, one can find out whether one of the two is better for smaller/bigger values for α. In general, as already mentioned in this section, other comparisons can be made, such as the composition of the instances or the value of B.

References

[1] Alexander Schrijver. On the history of combinatorial optimization (till 1960). *Handbooks in operations research and management science*, 12:38–46, 2005.

[2] Gregory Gutin and Abraham P Punnen. *The traveling salesman problem and its variations*, volume 12. Springer Science & Business Media, 2006.

[3] Richard M Karp. Reducibility among combinatorial problems. In *Complexity of computer computations*, pages 85–103. Springer, 1972.

[4] Tu-San Pham, Pieter Leyman, and Patrick De Causmaecker. The intermittent travelling salesman problem. *International Transactions in Operational Research*, pages 1–2, 2015.

[5] E Wacholder, J Han, and RC Mann. A neural network algorithm for the multiple traveling salesmen problem. *Biological Cybernetics*, 61(1):11–19, 1989.

[6] Yvan Dumas, Jacques Desrosiers, Eric Gelinas, and Marius M Solomon. An optimal algorithm for the traveling salesman problem with time windows. *Operations research*, 43(2):367–371, 1995.

[7] Norbert Ascheuer, Matteo Fischetti, and Martin Grötschel. A polyhedral study of the asymmetric traveling salesman problem with time windows. *Networks: An International Journal*, 36(2):69–79, 2000.

[8] Tu-San Pham, Pieter Leyman, and Patrick De Causmaecker. The intermittent travelling salesman problem. *International Transactions in Operational Research*, pages 4–5, 2015. Chapter 3.

[9] Vijay V Vazirani. *Approximation algorithms*. Springer Science & Business Media, 2013.

A Results method comparison

Nodes	EOCT Method 1	EOCT Method 2
2	0,9369623569	0,959015749
3	0,984783133	0,9851670939
4	0,9937512218	0,994803804
5	0,9994339623	0,998810946
6	0,9999126174	0,9992267431
7	0,9999996382	0,9999629117
8	1	0,9999993771
9	1	1
10	1	1

Table 3: The average of the EOCT of 100 random graphs applied to both methods. $\alpha = 1$*,* a_u *uniform from* $[1, 30]$ *and* B *uniform from* $[4, 10]$*. Software: AIMMS. Processor: Intel Core i7-6700HQ Processor (6MB Cache, up to 3.50GHz)*

B AIMMS code

```
## ams_version=1.0

Model Main_ITSP{
    Procedure SolveInstance {
        Body: {
            solve Solution;
            OptimalOrder(u,t) := ProcessingTime(u,t);
        }
    }
    Procedure SolveAmongAllPosibilities {
        Body: {
            solve Solution;
            OptimalCompletionTime := TotalCompletionTime;
            OptimalAmountOfTimeSteps := AmountOfTimeSteps;
            for ( n in ForLoop) do
            AmountOfTimeSteps := n;
            solve Solution;
            if (TotalCompletionTime < OptimalCompletionTime) then
             OptimalCompletionTime := TotalCompletionTime;
             OptimalAmountOfTimeSteps := n;
            endif;
            endfor;
            AmountOfTimeSteps := OptimalAmountOfTimeSteps;
            solve Solution;
        }
    }
    Procedure ClearAll {
        Body: {
            AmountOfTimeSteps := AmountOfNodes+1;
            empty OptimalCompletionTime, OptimalAmountOfTimeSteps, OptimalOrder,
            ProcessedOrNot, ProcessingTime, Traveling, WaitingTime, TimeBeginning,
            TemperatureAtTime, TotalCompletionTime;
        }
    }
    Procedure FindOptimumFast {
        Body: {
            empty OptimalCompletionTime, OptimalAmountOfTimeSteps, OptimalOrder,
            ProcessedOrNot, ProcessingTime, Traveling, WaitingTime, TimeBeginning,
            TemperatureAtTime, TotalCompletionTime;
            !AmountOfTimeSteps := max(AmountOfNodes + 1,ceil(sum[u, DrillingTime(u)/B]));
            AmountOfTimeSteps := AmountOfNodes + 1;
            solve Solution;
            while (AmountOfConsecutiveDrills = 0) do
             OptimalOrder(u,t) := ProcessingTime(u,t);
             OptimalCompletionTime := TotalCompletionTime;
             AmountOfTimeSteps +=2;
             solve Solution;
             if (AmountOfConsecutiveDrills = 2) then
             break;
             elseif (AmountOfConsecutiveDrills = 1) then
             AmountOfTimeSteps -= 1;
             solve Solution;
             OptimalOrder(u,t) := ProcessingTime(u,t);
             OptimalCompletionTime := TotalCompletionTime;
```

```
          break;
          endif;
         endwhile;
         if (AmountOfConsecutiveDrills <> 0) then
          AmountOfTimeSteps -= AmountOfConsecutiveDrills;
          solve Solution;
          OptimalOrder(u,t) := ProcessingTime(u,t);
          OptimalCompletionTime := TotalCompletionTime;
         endif;

         if(Solution.ProgramStatus <> 'Optimal') then
          empty OptimalCompletionTime, OptimalAmountOfTimeSteps, OptimalOrder,
          ProcessedOrNot, ProcessingTime, Traveling, WaitingTime, TimeBeginning,
          TemperatureAtTime, TotalCompletionTime;
         endif;
    }
}
Procedure BuildGraph {
    Body: {
         empty OptimalCompletionTime, OptimalAmountOfTimeSteps, OptimalOrder,
         ProcessedOrNot, ProcessingTime, Traveling, WaitingTime, TimeBeginning,
         TemperatureAtTime, TotalCompletionTime, ChosenPoints;
         while (card(ChosenPoints) <> AmountOfNodes) do
           empty Chosenpoints;
          for (u in Nodes) do
          ChosenPoints += Floor(Uniform(0,100));
          DrillingTime(u) := Uniform(0,50);
          endfor;
         endwhile;

         !for (u | DrillingTime(u) > B) do
         ! Check(u) := DrillingTime(u)-B;
         !endfor;

         B := Uniform(8,15);
         Distance(u,v) := Distances(u,v);
    }
}
Procedure BuildIntegerGraph {
    Body: {
         empty OptimalCompletionTime, OptimalAmountOfTimeSteps, OptimalOrder,
         ProcessedOrNot, ProcessingTime, Traveling, WaitingTime, TimeBeginning,
         TemperatureAtTime, TotalCompletionTime, ChosenPoints;
         while (card(ChosenPoints) <> AmountOfNodes) do
           empty Chosenpoints;
          for (u in Nodes) do
          ChosenPoints += round(Uniform(0,99));
          DrillingTime(u) := round(Uniform(1,20));
          endfor;
         endwhile;

         B := round(Uniform(4,10));
         Distance(u,v) := round(Distances(u,v));
    }
}
Procedure SameDistances {
```

```
    Body: {
        empty OptimalCompletionTime, OptimalAmountOfTimeSteps, OptimalOrder,
        ProcessedOrNot, ProcessingTime, Traveling, WaitingTime, TimeBeginning,
        TemperatureAtTime, TotalCompletionTime, ChosenPoints;
        while (card(ChosenPoints) <> AmountOfNodes) do
          empty Chosenpoints;
         for (u in Nodes) do
         ChosenPoints += round(Uniform(0,99));
         endfor;
        endwhile;
        Distance(u,v) := Dist;
        Distance(u,u):=0;
    }
}
Procedure CheckDistances {
    Body: {
        empty OptimalCompletionTime, OptimalAmountOfTimeSteps, OptimalOrder,
        ProcessedOrNot, ProcessingTime, Traveling, WaitingTime, TimeBeginning,
        TemperatureAtTime, TotalCompletionTime, ChosenPoints;
        while (card(ChosenPoints) <> AmountOfNodes) do
          empty Chosenpoints;
         for (u in Nodes) do
         ChosenPoints += round(Uniform(0,99));
         DrillingTime(u) := round(Uniform(1,50));
         endfor;
        endwhile;

        DrillingTime(1) := 0;
        B := round(Uniform(8,15));
        Distance(u,v) := round(Distances(u,v));

        AmountOfTimeSteps := AmountOfNodes + 1;
        solve Solution;
    }
}
Procedure NoDistSolve {
    Body: {
        empty StartJob, TotalCompletionTime2;
        Job(u,i):=max(0,ceil(min(B, DrillingTime(u)-(i-1)*B)));
        solve DOSolve;
    }
}
Procedure NoDistSolve2 {
    Body: {
        empty StartJob, TotalCompletionTime2;
        Job(u,i):=max(0,min(ceil(DrillingTime(u)/ceil(DrillingTime(u)/B)),
        DrillingTime(u)-(i-1)*ceil(DrillingTime(u)/ceil(DrillingTime(u)/B))));
        solve DOSolve;
    }
}
Procedure Procedure_1 {
    Body: {
        empty OptimalCompletionTime, OptimalAmountOfTimeSteps, OptimalOrder,
        ProcessedOrNot, ProcessingTime, Traveling, WaitingTime, TimeBeginning,
        TemperatureAtTime, TotalCompletionTime, ChosenPoints,
        StartJob, TotalCompletionTime2, OutcomeTest;
```

```
        for (x in Tests) do
        empty OptimalCompletionTime, OptimalAmountOfTimeSteps, OptimalOrder,
        ProcessedOrNot, ProcessingTime, Traveling, WaitingTime, TimeBeginning,
        TemperatureAtTime, TotalCompletionTime, ChosenPoints;
        while (card(ChosenPoints) <> AmountOfNodes) do
          empty Chosenpoints;
         for (u in Nodes) do
         ChosenPoints += round(Uniform(0,99));
         DrillingTime(u) := round(Uniform(1,30));
         endfor;
        endwhile;

        B := round(Uniform(4,10));
        Distance(u,v) := round(Distances(u,v));

        Job(u,i):=max(0,ceil(min(B, DrillingTime(u)-(i-1)*B)));
        solve DOSolve;
        OutcomeTest(x,1) := EXPTotalDrillingTime/TotalCompletionTime2;

        Job(u,i):=max(0,min(ceil(DrillingTime(u)/ceil(DrillingTime(u)/B)),
        DrillingTime(u)-(i-1)*ceil(DrillingTime(u)/ceil(DrillingTime(u)/B))));
        solve DOSolve;
        OutcomeTest(x,2) := EXPTotalDrillingTime/TotalCompletionTime2;
        endfor;
    }
}
DeclarationSection Make_Instance {
    ElementParameter SetColor {
        IndexDomain: p;
        Range: AllColors;
        Definition: {
            if (p in ChosenPoints) then 'black' else 'light grey' endif;
        }
    }
    Parameter NrColumns {
        Range: integer;
        Definition: 10;
    }
    Parameter NrRows {
        Range: integer;
        Definition: 10;
    }
    Parameter NrPoints {
        Range: integer;
        Definition: NrColumns * NrRows;
    }
    Parameter RowNr {
        IndexDomain: p;
        Range: integer;
        Definition: (p - ColumnNr(p))/NrColumns;
    }
    Parameter ColumnNr {
        IndexDomain: p;
        Range: integer;
        Definition: Mod(p, NrColumns);
    }
```

```
    Set Grid {
        SubsetOf: Integers;
        Index: p, q;
        Property: ElementsAreNumerical;
        Definition: {
            {0..NrPoints-1}
        }
    }
    Set ChosenPoints {
        SubsetOf: Grid;
    }
    Parameter WhichPoints {
        IndexDomain: u;
        Range: integer;
        Definition: Element(ChosenPoints,u);
    }
    Parameter Delta {
        Definition: 1;
    }
    Parameter SameDrilling {
        Range: integer;
    }
    Parameter Distances {
        IndexDomain: (u,v);
        Range: nonnegative;
        Definition: sqrt(sqr((Rownr(WhichPoints(u))-RowNr(WhichPoints(v)))*Delta)+
        sqr((ColumnNr(WhichPoints(u))-ColumnNr(WhichPoints(v)))*Delta));
    }
    Parameter Edges {
        IndexDomain: (p,q);
        Range: binary;
        Definition: {
            if (p in ChosenPoints AND q in ChosenPoints) then 1 else 0 endif;
        }
    }
}
Set Nodes {
    SubsetOf: Integers;
    Index: u, v, w;
    Property: ElementsAreNumerical;
    Definition: {
        {1..AmountOfNodes}
    }
}
Set ForLoop {
    SubsetOf: Integers;
    Index: n;
    Definition: {
        {AmountOfNodes+2..sum[u, DrillingTime(u)]}
    }
}
Set TimeSteps {
    SubsetOf: Integers;
    Index: t, z;
    Property: ElementsAreNumerical;
    Definition: {
```

```
        {1..AmountOfTimeSteps}
    }
}
Set TotalConstraints {
    SubsetOf: AllConstraints;
    Definition: {
        data
            {OneProcessedAtTime1, OneProcessedAtTime2, OneProcessedAtTime3,
            EnoughDrilling, CorrectWaitingTime, NotOverMaximum1,
            Startpoint, Endpoint, NotOverMaximum2, CorrectArrivalTime,
            CorrectArrivalTemperature}
    }
}
Parameter B {
    Range: nonnegative;
}
Parameter AmountOfNodes {
    Range: integer;
    InitialData: 4;
}
Parameter AmountOfTimeSteps {
    Range: integer;
    InitialData: 0;
}
Parameter Dist {
    InitialData: 10;
}
Parameter DrillingTime {
    IndexDomain: u;
    Range: nonnegative;
}
Parameter Distance {
    IndexDomain: (u,v);
    Range: nonnegative;
}
Parameter OptimalAmountOfTimeSteps {
    Range: integer;
}
Parameter OptimalCompletionTime {
    Range: nonnegative;
    InitialData: INF;
}
Parameter ConsecutiveDrills {
    IndexDomain: (u,t) | ProcessedOrNot(u,t)=1 AND t <> 1;
    Range: binary;
    Definition: {
        if (ProcessedOrNot(u,t-1)=1) then 1 endif;
    }
}
Parameter AmountOfConsecutiveDrills {
    Range: integer;
    Definition: sum[(u,t), ConsecutiveDrills(u,t)];
}
Parameter OptimalOrder {
    IndexDomain: (u,t);
    Range: nonnegative;
```

```
}
Parameter Difference {
    IndexDomain: (u,v,w) | w <> u AND u <> v AND DrillingTime(u) > B;
    Definition: {
        !Distance(u,w)+Distance(w,v)-Distance(u,v)
        Distance(u,v) + Distance(u,w) - Distance(v,w) - min(DrillingTime(u)-B,B)
    }
}
Parameter MinDifference {
    Range: free;
    Definition: min((u,v,w), Difference(u,v,w));
}
Parameter Alpha {
    InitialData: 1;
}
Parameter EXPTotalDrillingTime {
    Range: nonnegative;
    Definition: max(((Alpha+1)*max(u, DrillingTime(u))-B)/Alpha,
    sum[u, DrillingTime(u)]);
}
Variable ProcessedOrNot {
    IndexDomain: (u,t);
    Range: binary;
}
Variable ProcessingTime {
    IndexDomain: (u,t);
    Range: nonnegative;
}
Variable Traveling {
    IndexDomain: (u,v,t) | t <> AmountOfTimeSteps;
    Range: binary;
}
Variable WaitingTime {
    IndexDomain: (u,t);
    Range: nonnegative;
}
Variable TimeBeginning {
    IndexDomain: t;
    Range: nonnegative;
}
Variable TemperatureAtTime {
    IndexDomain: (u,t);
    Range: nonnegative;
}
Variable TotalCompletionTime {
    Range: free;
    Definition: TimeBeginning(AmountOfTimeSteps) +
    sum[u, ProcessingTime(u, AmountOfTimeSteps)] +
    sum[u, WaitingTime(u,AmountOfTimeSteps)];
}
Constraint OneProcessedAtTime1 {
    IndexDomain: t;
    Definition: sum[u, ProcessedOrNot(u,t)]<=1;
}
Constraint OneProcessedAtTime2 {
    IndexDomain: (u,t) | t <> AmountOfTimeSteps;
```

```
    Definition: ProcessedOrNot(u,t) = sum[v, Traveling(u,v,t)];
}
Constraint OneProcessedAtTime3 {
    IndexDomain: (u,t) | t <> 1;
    Definition: ProcessedOrNot(u,t) = sum[v, Traveling(v,u,t-1)];
}
Constraint EnoughDrilling {
    IndexDomain: u;
    Definition: sum[t, ProcessingTime(u,t)]=DrillingTime(u);
}
Constraint Startpoint {
    Definition: {
        ProcessedOrNot(1,1)=1;
    }
}
Constraint Endpoint {
    Definition: {
        ProcessedOrNot(1,AmountOfTimeSteps)=1;
    }
}
Constraint CorrectWaitingTime {
    IndexDomain: (u,t);
    Definition: WaitingTime(u,t) <= ProcessedOrNot(u,t)*Max(DrillingTime(u))/Alpha;
}
Constraint NotOverMaximum1 {
    IndexDomain: (u,t);
    Definition: ProcessingTime(u,t) <= ProcessedOrNot(u,t)*B +
    Alpha*WaitingTime(u,t);
}
Constraint NotOverMaximum2 {
    IndexDomain: (u,t);
    Definition: TemperatureAtTime(u,t) + ProcessingTime(u,t) <= B +
    Alpha*WaitingTime(u,t);
}
Constraint CorrectArrivalTime {
    IndexDomain: t | t <> AmountOfTimeSteps;
    Definition: TimeBeginning(t+1) >= TimeBeginning(t) +
    sum[u, ProcessingTime(u,t)] + sum[(u,v), Distance(u,v) * Traveling(u,v,t)] +
    sum[u, WaitingTime(u,t)];
}
Constraint CorrectArrivalTemperature {
    IndexDomain: (u,t) | t <> AmountOfTimeSteps;
    Definition: {
        TemperatureAtTime(u,t+1) >= TemperatureAtTime(u,t) +
        ProcessingTime(u,t) - Alpha*(TimeBeginning(t+1)-
        (TimeBeginning(t)+ ProcessingTime(u,t)))! + WaitingTime(u,t)
    }
}
MathematicalProgram Solution {
    Objective: TotalCompletionTime;
    Direction: minimize;
    Constraints: TotalConstraints;
    Variables: AllVariables;
    Type: MIP;
}
DeclarationSection NoDistances {
```

```
Set NoDistConstraints {
    SubsetOf: AllConstraints;
    Definition: data {EnoughProcessed, OnlyOneAtTime, EnoughCooledDown,
    CorrectTotalTime};
}
Set SubDrilling {
    SubsetOf: Integers;
    Index: i;
    Property: ElementsAreNumerical;
    Definition: {
        {1..Max(u, ceil(DrillingTime(u)/B))}
    }
}
Set TimeSteps2 {
    SubsetOf: Integers;
    Index: k, l;
    Definition: {
        {0..round(max(((Alpha+1)*max(u, DrillingTime(u))-B)/Alpha,
        sum[u, DrillingTime(u)])+B)}
    }
}
Set TwoMethods {
    SubsetOf: Integers;
    Index: m;
    Property: ElementsAreNumerical;
    Definition: {
        {1,2}
    }
}
Set Tests {
    SubsetOf: Integers;
    Index: x;
    Property: ElementsAreNumerical;
    Definition: {
        {1..AmountOfTests}
    }
}
Parameter Job {
    IndexDomain: (u,i);
    Range: nonnegative;
}
Parameter AmountOfTests {
    Range: integer;
    Definition: 100;
}
Parameter OutcomeTest {
    IndexDomain: (x,m);
}
Parameter AveragePerMethod {
    IndexDomain: m;
    Definition: sum[x, OutcomeTest(x,m)]/AmountOfTests;
}
Variable StartJob {
    IndexDomain: (u,i,k) | Job(u,i) > 0;
    Range: binary;
}
```

```
    Variable TotalCompletionTime2 {
        Range: free;
    }
    Constraint EnoughProcessed {
        IndexDomain: (u,i) | Job(u,i) > 0;
        Definition: sum[k, StartJob(u,i,k)]=1;
    }
    Constraint OnlyOneAtTime {
        IndexDomain: k;
        Definition: sum[(u,i), sum[l | k-Job(u,i) <= l <= k-1, StartJob(u,i,l)]]
        <= 1;
    }
    Constraint EnoughCooledDown {
        IndexDomain: (u,i,k) | Job(u,i) > 0 AND i > 1;
        Definition: StartJob(u,i,k) <= sum[l | l <= k-Job(u,i-1)-
        ceil(Job(u,i-1)/Alpha), StartJob(u,i-1,l)];
    }
    Constraint CorrectTotalTime {
        IndexDomain: (u,i,k) | Job(u,i) > 0;
        Definition: TotalCompletionTime2 >= k*StartJob(u,i,k) + Job(u,i);
    }
    MathematicalProgram DOSolve {
        Objective: TotalCompletionTime2;
        Direction: minimize;
        Constraints: NoDistConstraints;
        Variables: AllVariables;
        Type: Automatic;
    }
}
Procedure MainInitialization {
    Comment: "Add initialization statements here that do NOT require any
    library being initialized already.";
}
Procedure PostMainInitialization {
    Comment: {
        "Add initialization statements here that require that the libraries
        are already initialized properly, or add statements that require the
        Data Management module to be initialized."
    }
}
Procedure MainExecution;
Procedure PreMainTermination {
    Body: {
        return DataManagementExit();
    }
    Comment: {
        "Add termination statements here that require all libraries to be
        still alive.
        Return 1 if you allow the termination sequence to continue.
        Return 0 if you want to cancel the termination sequence."
    }
}
Procedure MainTermination {
    Body: {
        return 1;
    }
```

```
        Comment: {
            "Add termination statements here that do not require all libraries to be
            still alive.
            Return 1 to allow the termination sequence to continue.
            Return 0 if you want to cancel the termination sequence.
            It is recommended to only use the procedure PreMainTermination to cancel
            the termination sequence and let this procedure always return 1."
        }
    }
}
```