Integrated Scheduling and Location Models: Single Machine Makespan Problems

Holger Hennes *　　Horst W. Hamacher *

May 2002

Report in Wirtschaftsmathematik. No. 82/2002.
University of Kaiserslautern.

Abstract

Scheduling and location models are often used to tackle problems in production, logistics, and supply chain management. Instead of treating these models independent of each other, as is usually done in the literature, we consider in this paper an integrated model in which the locations of machines define release times for jobs. Polynomial solution algorithms are presented for single machine problems in which the scheduling part can be solved by the earliest release time rule.

Key words. Location problems, scheduling theory.

1 Introduction

In this paper we combine single facility network location and single machine scheduling problems. Both fields are well known from literature, but they are usually studied separately from each other. Maybe because locational decisions are seen as strategic and scheduling decisions as operational decisions. But examples in industrial planning or computer science show that this is not necessarily true. The usage of movable machines is, for instance, an example where the scheduling as well as the location decision is an operational one. A simultaneous re-planning of machine location and reorganization of the production line is useful to increase productivity.

*{hennes, hamacher} @mathematik.uni-kl.de, Universität Kaiserslautern, Germany

In order to combine location and scheduling we consider a graph $G = (V, E)$ with node set $V = \{v_1, \dots, v_n\}$ and edge set $E = \{e_1, \dots, e_m\}$. The nodes are identified with jobs $A_1, \dots, A_n$, each with processing time $p_j, j = 1, \dots, n$. The edges $e = [v_i, v_j]$ represent the possibility of moving job A_i to job A_j (or vice versa) directly within $l_e = l_{ij}$ time units. If there exists no edge between A_i and A_j, we move the jobs along the shortest paths from v_i to v_j with respect to the edge lengths l_e. Using an all pair shortest path algorithm (see, e.g. Ahuja et al. [AMO93]) the shortest path distance d_{ij} between all nodes v_i and v_j in V can be computed in $O(n^3)$ time.

The location part of the problem is the question where to position one machine on which all jobs $A_1, \dots, A_n$ have to be processed. In the **node version** of the **Scheduling/Location (ScheLoc)problem**, the positioning is only allowed in the nodes while in the **absolute version of ScheLoc** the machine can be placed anywhere in the graph, i.e. also in the interior of an edge. If $e = [v_i, v_j] \in E$, then $x = (e, \alpha)$ with $\alpha \in [0, 1]$ is the **point** in G with

$$d(v_i, x) = d(x, v_i) = \alpha l_e \text{ and } d(v_j, x) = d(x, v_j) = (1 - \alpha) l_e \qquad (1)$$

(see Figure 1), i.e. we assume linearity of the distance along edge e. The set of points in G is denoted by $\mathcal{X}$.

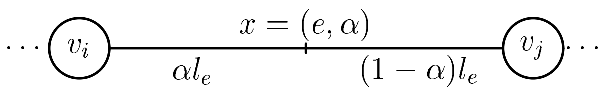

Figure 1: Representation of point $x = (e, \alpha)$ in the interior of edge $e = [v_i, v_j]$

In terms of the classification scheme proposed by Hamacher and Nickel [HN99] the problem is $1/G/./d(V, \{V, G\})/obj$, where the objective function obj is given by the scheduling problem. For an overview of location problems see e.g. Daskin [Das95], Drezner and Hamacher [DH02], Francis et al. [FMW92] or Mirchandani and Francis [MF90].

Once a position has been chosen for the machine, ScheLoc reduces to a classical scheduling problem with release dates. The machine can only process one of the jobs $A_1, \dots, A_n$ at a time. During the processing, preemption is not allowed. In order to transport any job A_k to position x of the machine it takes

$$d(v_k, x) = \begin{cases} d_{kl} & \text{if } x = v_l \in V, \\ min\{d(v_k, v_i) + \alpha l_{ij}, & \\ \qquad d(v_k, v_j) + (1 - \alpha) l_{ij}\} & \text{if } x = (e, \alpha) \in \mathcal{X} \setminus V,\ e = [v_i, v_j] \end{cases} \qquad (2)$$

time units, such that the processing of job A_k can not be started before its release time $r_k(x) = d(v_k, x)$.

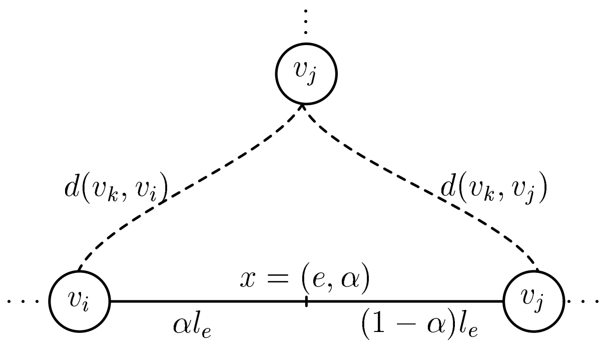

Figure 2: Visualization of release time $r_k(x) = d(v_k, x)$ for job A_k if machine is located in $x \in \mathcal{X}$

The largest part of the paper deals with the makespan objective, i.e. the maximal completion time of the jobs $C_{max}(x) := \max\{C_i(x), i \in \{1, \dots, n\}\}$, where $C_i(x)$ is the time when the processing of job A_i is completed. Any schedule is given by an ordering of the jobs, denoted by $\Pi = (\pi(1), \dots, \pi(n)$, i.e. $C_{max}(x) = C_{\pi(n)}(x))$. In the classification scheme of Graham et al. [GLLRK79] the problem is denoted $1 \mid r_i \mid C_{max}$. This problem can be solved by sorting the jobs in non-decreasing order of the their release dates (**earliest release date (ERD) rule**). An overview of scheduling theory is given e.g. in Baker [Bak74], Blazewicz et al. [BEP$^+$01], Brucker [Bru01], Coffman [Cof76], Conway et al. [CMM67], French [Fre82], Pinedo [Pin95] or Tanaev et al. [TSS94a] and [TSS94b].

In contrast to classical scheduling problems, the problem is dependent on the location x of the machine. In particular, the release dates are in ScheLoc not part of the input, but a consequence of the choice of the location x for the machine. ScheLoc can therefore be written as

$$\min_{x \in \mathcal{X}} C_{max}(x) = \max_{j=1}^{n} C_j(x) \tag{3}$$

In the next section we will show how to solve the simplest version of ScheLoc involving just two jobs A_1 and A_2 and a single edge graph $G = (\{v_1, v_2\}, [v_1, v_2])$. The insight obtained from solving this problem will be used in Section 3 to solve, in sequence, the node version of ScheLoc and the absolute version ScheLoc problem in trees and in general graphs. Algorithms are presented to solve the problems. We conclude the paper by observing that the approach described in this paper carries over to all scheduling problems which can be solved by the ERD rule.

2 Two Job ScheLoc

Given are two jobs A_1 and A_2 with their processing times p_1 and p_2 and their representation as nodes v_1 and v_2, respectively. We denote the distance between the two locations of the jobs by

$$D := d(v_1, v_2). \tag{4}$$

With $x_\alpha := ([v_1, v_2], \alpha)$ we denote the point with distance

$$d(v_1, x_\alpha) = \alpha D \tag{5}$$

from v_1 and distance

$$d(v_2, x_\alpha) = (1 - \alpha)D \tag{6}$$

from v_2, where $\alpha \in [0, 1]$.

The position $x_{0.5}$ is given as separation point because

$$r_1(x_{0.5}) = d(v_1, x_{0.5}) = d(x_{0.5}, v_2) = r_2(x_{0.5}), \tag{7}$$

i.e. $r_1(x) < r_2(x)$ holds for $x \in (v_1, x_{0.5})$, and $r_1(x) > r_2(x)$ holds for $x \in (x_{0.5}, v_2)$. Consequently, the ERD rule defines two regions

$$[v_1, x_{0.5}],\ [x_{0.5}, v_2]. \tag{8}$$

on the edge, in which the processing sequence is fixed, namely $\pi = \{1, 2\}$ in $[v_1, x_{0.5}]$ and $\pi = \{2, 1\}$ in $[x_{0.5}, v_2]$. These regions, in which the job sequence is not changing are called ordered Weber regions (see Nickel and Puerto, [NP99]) and will play an important role in Section 3.2.

The objective value $z(x)$ is equal to the completion time of the second processed job, i.e.

$$z(x) = \max\{C_1(x), C_2(x)\} = C_{\pi(2)}(x). \tag{9}$$

Case 1: $\pi(1) = 1$ and $\pi(2) = 2$, i.e. we locate the machine in $x_\alpha \in [v_1, x_{0.5}]$ and A_1 is processed first. In this case the objective value is

$$\begin{aligned} z(x) = C_2(x) &= \max\{C_1(x) + p_2,\ r_2(x) + p_2\} && (10)\\ &= \max\{r_1(x) + p_1 + p_2, r_2(x)p_2\}, && (11) \end{aligned}$$

since A_2 can not start its processing before job A_1 is completed and, for sure, not before its release date.

Since $r_1(x_\alpha) = \alpha D$ is increasing and $r_2(x_\alpha) = (1 - \alpha)D$ is decreasing for $\alpha \in [0, 0.5]$ we get the minimal value of $z(x_\alpha)$ for α in the intersection point of $\alpha D + p_1 + p_2$ and $(1 - \alpha)D + p_2$.

Lemma 2.1 *For job sequence $\pi(1) = 1$, $\pi(2) = 2$, the objective value achieves its minimum, if and only if*

$$r_1(x_\alpha) + p_1 = c_1(x_\alpha) = r_2(x_\alpha), \tag{12}$$

i.e. the optimal location x_α has to be chosen such that

$$\alpha = \alpha_{12} = \frac{D - p_1}{2D}. \tag{13}$$

The optimal objective value is

$$z(x_{\alpha_{12}}) = \frac{D - p_1}{2} + p_2 \tag{14}$$

<u>Case 2:</u> $\pi(1) = 2, \pi(2) = 1$, i.e. we locate the machine in $x_\alpha \in [x_{0.5}, v_2]$ and A_2 is processed first. As in Case 1 we obtain for x_α:

$$\begin{aligned} z(x_\alpha) = C_1(x_\alpha) &= \max\{C_2(x_\alpha) + p_1, r_1(x_\alpha) + p1\} && (15)\\ &= \max\{(1-\alpha)D + p_2 + p_1, \alpha D + p_1\}, && (16) \end{aligned}$$

such that we can conclude.

Lemma 2.2 *For job sequence $\pi(1) = 2, \pi(2) = 1$ the optimal objective value is*

$$z(x_{\alpha_{21}}) = \frac{D + p_2}{2} + p_1, \tag{17}$$

where $\alpha_{21} = \frac{D+p_2}{2D}$.

By comparing $z(x_{\alpha_{12}})$ and $z(x_{\alpha_{21}})$ we obtain a closed form expression for the two job optimal solution.

Lemma 2.3 *The optimal location x_{α^*} for the machine in the single machine ScheLoc problem with 2 jobs is*

$$\alpha^* = \begin{cases} \frac{D-p_1}{2D} & \textit{if } p_1 \geq p_2 \\ \frac{D+p_2}{2D} & \textit{if } p_1 \leq p_2. \end{cases} \tag{18}$$

A visualization of the results in this section is depicted in Figure 3.

3 ScheLoc with more than Two Jobs

If the machine is to be positioned on a graph there are two alternatives. Locating is only possible in nodes or we allow that the machine can also be positioned inside the edges, i.e. locating is possible on the whole graph. For the second possibility we consider first the special case that the graph is a tree and then we generalize the results to arbitrary graphs.

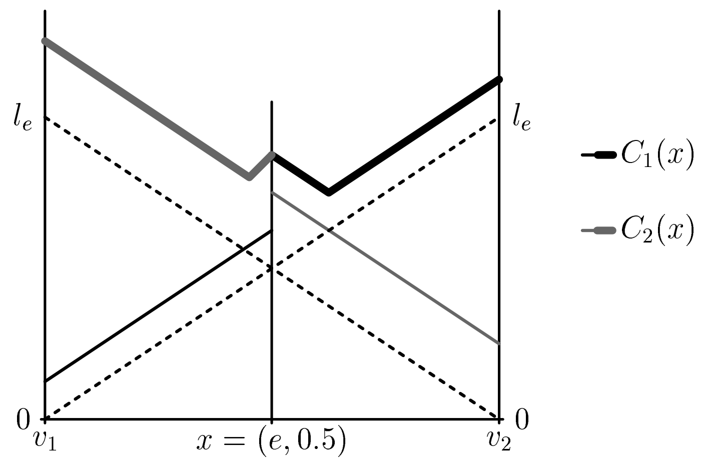

Figure 3: Visualization of release dates(doted lines), completion times(full lines) and C_{max}(thick lines) depending on x

3.1 Node Version of ScheLoc

This problem can be solved by complete enumeration. Every node is considered to be a possible location of the machine. After fixing the position of the machine we can calculate the release dates and the solution of the scheduling problem in this node. The best solution gives us the optimal node position of the machine.

Since the time complexity for the scheduling part is $O(n \log n)$ in every node the overall complexity is $O(n^2 \log n)$. Note that this complexity is in general graphs dominated by the $O(n^3)$ complexity to find the distances d_{ij} by an all pair shortest path algorithm (see e.g. Ahuja et al. [AMO93]).

To make the calculation faster we use two kinds of lower bounds, an overall lower bound and one for a fixed position in a node. An overall lower bound is given by the sum of the processing times,

$$LB_1 := \sum_{i=1}^{n} p_i. \tag{19}$$

If the objective function in a node is equal to this lower bound we can stop the algorithm. This observation allows us to easily find the optimal position of the machine in a special case.

Lemma 3.1 *If a job A_i exists with $p_i \geq d(v_i, v_j)\ \forall j \in \{1, \dots, n\}$ then v_i is the optimal position of the machine.*

Proof. If $p_i \geq d(v_i, v_j)$ then $p_i \geq r_j(v_i), \forall j \in \{1, \dots, n\}$ and therefore $C_{max}(v_i) = \sum_{j=1}^{n} p_j = LB_1$, since $r_i(v_i) = 0$ and all jobs can be processed after A_i without idle time. ■

The local lower bound is given by the distance between a node v_j and the location of the machine plus the processing time of the corresponding job, i.e.

$$LB_i = \max_{j=1}^{n}\{d(v_i, v_j) + p_j\}. \tag{20}$$

If LB_i for a node v_i is greater than the value of the best solution found so far we can skip the calculation of the sequence in this node.

Due to the first lower bound sorting the jobs in non increasing order of their processing times (LPT rule) and search in this order for the optimal position yields good results for dense graphs like complete graphs and when the processing times are high compared with the distances between the nodes. If the graph is not dense, in particular if it is a tree we choose another strategy to search the optimal position. We sort the jobs in non decreasing order of the value LB_i. This strategy yields also good results if the graph is dense, but the processing times are low relative to the distances. Example 3.1 shows, however, that neither $argmax_{v_i \in V}\{p_i\}$ nor $argmin_{v_i \in V}\{LB_i\}$ gives the optimal position for the machine.

Example 3.1 *Let $n = 4$ and $p_1 = 10, p_2 = 1, p_3 = 9, p_4 = 8$, and let $G = (V, E)$ be the graph of Figure 4*

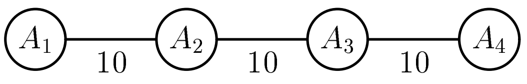

Figure 4: Example graph

$argmax_{v_i \in V}\{p_i\} = v_1$ and $argmin_{v_i \in V}\{LB_i\} = v_2$ but the optimal position is v_3(see Figure 5).

3.2 Absolute ScheLoc Problems in Trees

Given an undirected tree graph $G = (V, E)$. It is possible to locate the machine anywhere on the edges. We can use the concept of ordered Weber regions OWR (see Nickel and Puerto [NP99]) for solving this problem. A subset $R \subseteq \mathcal{X}$ of points is called **ordered Weber region (OWR)** if

$$d(x, v_i) \leq d(x, v_j) \Longleftrightarrow d(y, v_i) \leq d(y, v_j)\ \forall x, y \in R, v_i, v_j \in V. \tag{21}$$

Hence the list of distances $\{d(x, v_i) : v_i \in V\}$ is sorted in the same way for all $x \in R$. Consequently, the optimal job sequence is identical for all $x \in R$, since it is defined by the earliest release date rule and $r_i(x) = d(x, v_i)\ \forall v_i \in V$. In order to simplify the subsequent arguments we assume without loss of generality that

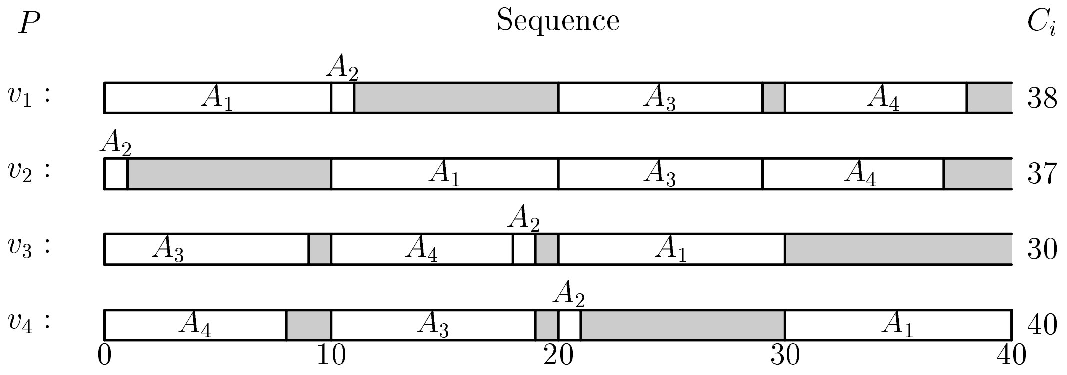

Figure 5: Gantt charts for the position of the machine in the nodes of Example 3.1

any OWR is a subset of a single edge e. That is $R = \{(e, \alpha) : \underline{\alpha} \leq \alpha \leq \overline{\alpha}\}$ for some $0 \leq \underline{\alpha} \leq \overline{\alpha} \leq 1$. (If this is not the case, i.e. a node lies inside an OWR, we decompose it into several parts separated by this node.)

Since in an ordered Weber region R the order is not changing we can easily calculate the optimal position of the machine in R. The completion time of the whole processing is equal to the completion time of the last job

$$z(\pi) = c_{\pi(n)}(x) = \max\{c_{\pi(n-1)}(x), r_{\pi(n)}(x)\} + p_{\pi(n)},\ \forall x = (e, \alpha) \in R. \tag{22}$$

In this manner we can calculate the completion times of all jobs by

$$C_{\pi(i)}(x) = \max\{C_{\pi(i-1)}(x), r_{\pi(i)}(x)\} + p_{\pi(i)}\ \forall i = 2, \ldots, n,\ \forall x \in R, \tag{23}$$

with $C_{\pi(1)}(x) = r_{\pi(1)}(x) + p_{\pi(1)}$.

Since G is a tree any two points are connected by a unique path. Hence $r_i(x) = d(x, v_i)$ defined by (2) is linear in α for $x = (e, \alpha) \in R$ with slope ± 1.

Lemma 3.2 *The objective function is piecewise linear and convex in every OWR, consisting of at most two linear pieces with slopes* ± 1.

So we can calculate the optimal position of the machine in a given OWR in $O(n)$ as follows. First we compare $C_{\pi(n)}(e, \underline{\alpha})$ and $C_{\pi(n)}(e, \overline{\alpha})$ in the two boundary points of R. If

$$\mid C_{\pi(n)}(e, \underline{\alpha}) - C_{\pi(n)}(e, \overline{\alpha}) \mid = d(R) = (\overline{\alpha} - \underline{\alpha}) l_e, \tag{24}$$

the optimal position in R is the boundary point with the lower completion time. This situation occurs if $z(\pi)$ is linear. Otherwise, i.e. $z(\pi)$ consists of two linear pieces, Equation (24) does not hold and d_1 and d_2 solving

$$\begin{aligned} d_1 + d_2 &= d(R) \\ d_1 - d_2 &= C_{\pi(n)}(e, \underline{\alpha}) - C_{\pi(n)}(e, \overline{\alpha}), \end{aligned} \tag{25}$$

are the distances between the optimal location and the boundary point $(e, \underline{\alpha})$ and $(e, \overline{\alpha})$, respectively.

Now we can formulate the algorithm.

Algorithm 1 Locating a single machine on a tree network

```
Step 0: Calculate shortest paths of all pair of nodes;
Step 1: Identify the OWRs, i.e. regions with equal order of the jobs;
Step 2: For each region calculate the optimal machine location using (24) or (25);
Step 3: Take the best one as optimum;
```

In a tree every node $v_i \in E$ is connected to every other node $v_j \in E, i \neq j$ by a unique path P_{ij}. Every path has an unique midpoint dividing the path into two parts where every point on one side is nearer to v_i than to v_j and on the other side every point has a shorter distance to v_j than to v_i. The two parts of the graph belong therefore to different OWRs. In the graph we have $\frac{n(n-1)}{2}$ different pair of nodes and therefore $\frac{n(n-1)}{2}$ different paths. By constructing the OWRs by considering one path after the other we get $\frac{n(n-1)}{2} + 1$ OWRs. The first midpoint decomposes the tree into two regions. The second midpoint decomposes one of the two regions into two new regions and therefore we have three regions. Every other midpoint decomposes one existing region into two regions and the number of regions increases by one. If an OWR contains a node we decompose it again in parts that are on different edges, i.e. we decompose an OWR R into $\delta(v)$ parts, where $\delta(v)$ is the degree of node $v \in V$, if $v \in int(R)$. Since leaves have a node degree of one they do not separate an OWR. Degree $\delta(v) = 2$ leads to one additional OWR, degree $\delta(v) = 3$ to two additional OWRs, and so on. We get

$$\sum_{v \in V} (\delta(v) - 1) =\mid V \mid - 2 \tag{26}$$

additional OWRs. Thus the complete number of OWRs is

$$\frac{n(n-1)}{2} + n - 1 = \frac{n(n+1)}{2} + 1 = O(n^2). \tag{27}$$

Thus we have $O(n^2)$ OWRs in a tree. In every region we have to calculate the sequence which takes $O(n \log n)$ such that we have $O(n^3 \log n)$ for the whole algorithm.

To improve the time bound we construct the OWRs in a different way. We know that in the sequence moving over the boundary from one OWR to a neighbor OWR only those jobs are exchanged with equal distance to that boundary. If we know the sequence in one OWR we can easily calculate the sequence in the neighbor OWR and if we know the release times in one boundary of the OWR we can easily calculate the release times in the other boundary of the OWR. Starting in a boundary point of an OWR R and scanning through the tree along R the other boundary is reached if we reach a node $x = v_{\pi(1)}$, i.e. $r_{\pi(1)}(x) = 0$, where Π is the optimal sequence in R, or we reach the midpoint of a path between two nodes. Since the two jobs associated with the two nodes have to be successors in the sequence a midpoint is found if $r_{\pi(i)}(x) = r_{\pi(i+1)}(x)$, for some $i \in \{1, \dots, n\}$. The sequence is changed by swapping the two nodes $A_{\pi(i)}$ and $A_{\pi(i+1)}$. As starting point for the scan procedure a leaf is a good choice, since we know the distances to the other nodes and therefore the sequence. Additionally a leaf is always a boundary of an OWR. If the scan procedure reaches a node v with degree $\delta(v) > 2$ we store the information about the release times in that node and scan the rest of the tree branch by branch.

In the starting point we have to calculate the sequence which takes $O(n \log n)$ time. Scanning through an OWR takes $O(n)$ to follow the n distance functions and the exchange from one OWR to another has the same complexity, since we have to exchange at most $n/2$ jobs in the sequence. With the number of OWRs we have calculated before we get a complexity of $O(n^3)$ for implementing the algorithm.

3.3 ScheLoc in General Graphs

The algorithm of Section 3.2 for trees is based on the fact, that $r_k(x) = d(x, v_k)$ are linear functions in $x = (e, \alpha)$ on each OWR $R = \{(e, \alpha) : \underline{\alpha} \leq \alpha \leq \overline{\alpha}\}$ for all $v_k \in V$. In general graphs this is no longer true as can be seen from the definition of the distance function in (2).

Next we will show how to decompose OWRs for general graphs further such that the linearity property holds in the decomposition parts. For this purpose let R be an OWR. A point $x = (e, \alpha) \in R$ on edge $e = [v_i, v_j]$ is called **equilibrium point** if

$$d(v_i, v_k) + \alpha l_e = d(v_j, v_k) + (1 - \alpha) l_e \tag{28}$$

holds for some $v_k \in V$. Each edge - and thus each OWR - contains at most $(n - 2)$ equilibrium points. Notice that the linearity of $r_k(x)$ is violated in OWR R if and only if the equilibrium point with respect to node v_k is contained in the interior of R. Therefore if $\alpha_1, \dots, \alpha_H$ define the equilibrium points (e, α_h) in R, all functions $r_k(x)$ are linear in each of the segments

$$R_k := \{(e, \alpha) : \alpha_h \leq \alpha \leq \alpha_{h+1}\} \; \forall \; h = 1, \dots, H - 1. \tag{29}$$

Since $H \leq n - 2$ and there are $O(n^2)$ intersection points of two distance functions along one edge, we get at most $O(n^4)$ of such sets the union of which is $\mathcal{X}$.

The arguments used in the derivation of Algorithm 1 for trees now carry over to the partition of $\mathcal{X}$ into sets R_h. Since $r_k(x)$ is linear on R_h for all h, the objective function is convex on R_h and attains its optimal location either in one of its two boundaries (see Equation (24)) or can be computed using (25). The complexity of this algorithm is $O(n^5)$.

4 Extension to Other Scheduling and Location Problems

In the previous sections we have considered ScheLoc problems where the scheduling is a makespan problem. The crucial argument in the solution algorithms was that we can find a partition of the point set $\mathcal{X}$ into subsets in which the earliest release dates (ERD) sequence is unchanged and in which the release functions $r_k(x)$ are linear. This argument is, however independent of the objective function. Therefore the tools developed above apply to all ScheLoc problems where the scheduling part can be solved by the ERD rule. The list of problems include

$$1 \mid r_i \geq 0, d_i = d \, \forall i \in \{1, \dots, n\} \mid L_{max}$$

$$1 \mid r_i \geq 0, d_i = d \, \forall i \in \{1, \dots, n\}, prec \mid L_{max}$$

$$1 \mid r_i \geq 0, prec \mid C_{max}.$$

For the problems with precedence constraints we have to modify the release dates before starting our algorithms, such that $r_i \geq r_j$ if the job A_i is a successor of job A_j, i.e. $A_j \rightarrow A_i$. This can be done in $O(n + \epsilon)$, see [Bru01], where ϵ is the number of precedence constraints.

A similar approach can be used if we modify the location environment. If we consider location problems in the plane instead of networks, the concept of ordered Weber regions can be defined in this context as well. Correspondingly, the job sequences are identical for all points in the OWR and algorithms analogous to the ones developed in Section 3 can be derived.

References

[AMO93] R.K. Ahuja, T.L. Magnanti, and J.B. Orlin. *Network Flows - Theory, Algorithms, and Applications.* Prentice Hall, Englewood Cliffs, New Jersey, 1993.

[Bak74] K.R. Baker. *Introduction to Sequenzing and Scheduling.* John Wiley & Sons, 1974.

[BEP$^+$01] J. Blazewicz, K.H. Ecker, E. Pesch, G. Schmidt, and J. Weglarz. *Scheduling Computer and Manufacturing Processes.* Springer Verlag, Berlin, 2nd edition, 2001.

[Bru01] P. Brucker. *Scheduling Algorithms.* Springer Verlag, Berlin, 3rd edition, 2001.

[CMM67] R.W. Conway, W.L. Maxwell, and L.W. Miller. *Theory of Scheduling.* Addison-Wesley, 1967.

[Cof76] E.G. Jr. Coffman. *Computer and Job-Shop Scheduling.* John Wiley & Sons, 1976.

[Das95] M.S. Daskin. *Network and Discrete Location. Models, Algorithms and Applications.* Wiley-Interscene Series in Discrete Mathematics and Optimization. John Wiley and Sons, Inc., New York, NY, 1995.

[DH02] Z. Drezner and H.W. Hamacher, editors. *Facility Location, Applications and Theory.* Springer Verlag, Berlin, 2002.

[FMW92] R.L. Francis, L.F. Jr. McGinnis, and J.A. White. *Facility Layout and Location: An Analytical Approach.* Prentice Hall International Series in Industrial and Systems Engineering. Prentice Hall, Englewood Cliffs, New Jersey, 2nd edition, 1992.

[Fre82] S. French. *Sequenzing and Scheduling.* Ellis Horwood Series: Mathematics and its Applications. John Wiley & Son, 1982.

[GLLRK79] R.L. Graham, E.L. Lawler, J.K. Lenstra, and A.H.G. Rinnooy Kan. Optimization and approximation in deterministic sequenzing and scheduling: a survey. *Ann.discr.Math.*, 5:287–326, 1979.

[HN99] H.W. Hamacher and S. Nickel. Classification of location models. *Location Science*, 36:34–36, 1999.

[MF90] P.B. Mirchandani and R.L. Francis. *Discrete Location Theory.* Discrete Mathematics and Optimization. Wiley-Interscience Series, 1990.

[NP99] S. Nickel and J. Puerto. A unified approach to network location problems. *Networks*, 34(4):283–290, 1999.

[Pin95] M. Pinedo. *Scheduling: Theory, Algorithms and Systems.* Prentice Hall, 1995.

[TSS94a] V.S. Tanaev, Y.N. Sotskov, and V.A. Strusevitch. *Scheduling Theory: Multi-Stage Systems.* Kluver Academic Publishers, Dordrecht, 1994.

[TSS94b] V.S. Tanaev, Y.N. Sotskov, and V.A. Strusevitch. *Scheduling Theory: Single-Stage Systems.* Kluver Academic Publishers, Dordrecht, 1994.