Discrete Parallel Machine Makespan ScheLoc Problem

Corinna Heßler

Department of Mathematics,
TU Kaiserslautern, Germany,
c.hessler@mathematik.uni-kl.de

Kaouthar Deghdak

Laboratoire Informatique,
Université François-Rabelais de Tours, France,
deghdak@univ-tours.fr

27th July 2015

Scheduling-Location (ScheLoc) Problems integrate the separate fields of scheduling and location problems. In ScheLoc Problems the objective is to find locations for the machines and a schedule for each machine subject to some production and location constraints such that some scheduling objective is minimized. In this paper we consider the Discrete Parallel Machine Makespan (DPMM) ScheLoc Problem where the set of possible machine locations is discrete and a set of n jobs has to be taken to the machines and processed such that the makespan is minimized. Since the separate location and scheduling problem are both NP-hard, so is the corresponding ScheLoc Problem. Therefore, we propose an integer programming formulation and different versions of clustering heuristics, where jobs are split into clusters and each cluster is assigned to one of the possible machine locations. Since the IP formulation can only be solved for small scale instances we propose several lower bounds to measure the quality of the clustering heuristics. Extensive computational tests show the efficiency of the heuristics.

1 Introduction

Scheduling-Location (ScheLoc) Problems combine the two well-studied fields of location planning and scheduling theory. The goal of ScheLoc Problems is to simultaneously locate a set of machines and schedule a set of jobs on the machines such that the processing of the jobs is optimized. For processing on a machine the job has to be moved from its job location to the corresponding machine location. The point in time at which the job arrives at a machine is treated as the release date of the job. Since it is dependent on the distance from the job location to the machine location, the release-date of a job is dependent on the machine location. The objective of a ScheLoc Problem is always a pure scheduling objective, but due to the location-dependent release dates its value is also dependent on the location decisions.

While location planning is more of a strategic decision, machine scheduling is situated in the operational level of the supply chain. Still the two types of decisions depend largely on one-another and a sequential solution of the two problems will yield suboptimal results for the supply chain. There are many applications where simultaneous location and scheduling is required. One application mentioned in Kalsch [6] comes from the mining industry where crushers are used in processing minerals. Those crushers are movable and can change their location dependent on the set of valuable minerals to improve some scheduling objective. That means that for a given set of jobs we have to find a set of locations for the crushers such that a scheduling objective is optimized. Kalsch also presented various other applications.

To overcome the weaknesses of a sequential approach the ScheLoc Problem was introduced by Hennes [4] and Hennes and Hamacher [3]. They considered the Single Machine Network (SMN) ScheLoc Problem, i.e., the problem where the location of a single machine in a network is to be found, and reported some polynomial time algorithms for special cases. The Single Machine Planar (SMP) Makespan ScheLoc Problem was treated by Elvikis et al. [1] and Kalsch and Drezner [5]. Kalsch [6] considered ScheLoc Problems with a universal objective function that is defined similar to the Ordered Weber Problems in location theory (see e.g., Nickel and Puerto [10]) and integrates some well-known scheduling objectives as well as some new objective functions into a general framework. Some algorithms to solve special cases of the Single Machine Universal Planar (SMUP) ScheLoc Problem are presented and some special cases of the SMN ScheLoc Problem are considered. A Single Machine Universal Network (SMUN) ScheLoc Problem was treated in Kaufmann [7] and a polynomial time algorithm for an SMUN ScheLoc Problem with Preemption was proposed.

There are some problems in the literature that are related to ScheLoc problems as they combine scheduling problems with transportation decisions. However, most of those problems do not integrate the location decision. The only problem known to the authors that also considers location decisions was introduced in Wesolkowski et al. [13]. In this paper the problem of locating training devices for military forces such that the training can be completed at minimum cost is considered. The location part is the essential part of the problem as for each possible location the optimal mix of training devices has to be computed. The scheduling part is reduced to an assignment of soldiers to

locations. Furthermore, the problem is considered with multiple cost objectives. In the paper a multi-objective genetic algorithm is proposed that provides good results for the considered problem. But since the scheduling part is reduced to an assignment problem without accounting for any scheduling data (e.g., processing times) the genetic algorithm is not fit for ScheLoc Problems.

In this paper we will consider the Discrete Parallel Machine Makespan (DPMM) ScheLoc Problem, i.e., the problem of locating a set of parallel machines on a finite set of possible locations such that the given set of jobs are scheduled in such a way that the time at which the last job is completed is minimized. Parallel Machine ScheLoc Problems have been defined in Hennes [4] (for the network case) and Kalsch [6] (for the general case), but to our knowledge no work so far has dealt with the solution of this kind of problem.

The DPMM ScheLoc Problem consists of two problems: a discrete location problem and a parallel machine problem that are linked via the assignment of jobs to locations. Since both subproblems are $\mathcal{NP}$-hard, so is the ScheLoc Problem. If both the locations and the assignment are fixed the scheduling problem reduces to a polynomially solvable single machine problem for each location. Therefore, we focus on finding heuristics that solve the location and assignment problem taking into account the scheduling data and then optimally solve the reduced scheduling problem. Although the makespan objective is a maximum objective, it depends on the jobs assigned to each machine. Therefore, the location problem to be solved here is more similar to the p-median problem than to the p-center problem. There are several extensive surveys on algorithms for the p-median problem (e.g., Reese [11], Mladenović et al. [9], Tansel et al [12]).

A popular type of heuristic for the p-median problem are clustering heuristics, the first was proposed by Maranzana [8]. The clustering problem is very similar to the p-median problem and consists of splitting a set of objects into disjoint subsets based on similarity. In the location context the clustering problem is the problem of finding p cluster centers and an assignment of demand to clusters such that the sum of the distances of demand locations to the corresponding cluster centers is minimized. An extensive survey on general clustering algorithms is given in Xu and Wunsch [14]. This type of heuristic can be adapted to the ScheLoc context as the scheduling data can be integrated when computing cluster centers and the assignment and for each (partial) assignment an optimal schedule can be computed.

The paper is structered as follows: We will formally introduce the DPMM ScheLoc Problem in Section 2. In Section 3 we will give an IP-formulation and propose several lower bounds. We present some clustering type heuristics in Section 4 and a local search heuristic based on the clustering heuristics in Section 5. Computational results of all heuristics are provided in Section 6. We conclude with a summary and perspectives for future research in Section 7.

2 Problem Definition

We consider the Discrete Parallel Machine Makespan ScheLoc Problem. In this problem we choose p locations for machines out of a discrete set of possible locations $\mathcal{M} = \{1, \ldots, m\}$ and schedule a set of jobs $\mathcal{N} = \{1, \ldots, n\}$ on the parallel machines such that the makespan is minimized. Jobs are stored in so-called job locations and have to be moved to a machine for processing. Therefore, the earliest possible start time of a job, its release date, is dependent on the distance between the job and the machine location. Let $D \in \mathbb{R}^{n \times m}$ be the matrix of distances, i.e., $D(i,k) = dist(i,k)$ is the distance between the location of job $i \in \mathcal{N}$ and possible machine location $k \in \mathcal{M}$. There are several structures that may give rise to this distance matrix, e.g., shortest path distances where $\mathcal{N}$ and $\mathcal{M}$ are nodes in a graph or Euclidean distances where $\mathcal{N}$ and $\mathcal{M}$ are locations in the plane. The release date r_{ik} of job i if processed on a machine in location k is dependent on the storage arrival time σ_i, i.e., the time at which job i becomes available in its job location, the travel speed ν_{ik} which is the speed at which job i moves towards machine k, and the distance between the locations $dist(i,k)$. It is computed as follows

$$r_{ik} = \sigma_i + \tau_{ik} dist\left(a_i, v_k\right),$$

where $\tau_{ik} = 1/\nu_{ik}$. After arriving at a machine location a job has to be processed for p_i time units on the machine. Each machine can process at most one job at each point in time. The DPMM problem consists of selecting exactly p locations from $\mathcal{M}$ and scheduling all jobs $i \in \mathcal{N}$ on the selected machines such that the location-dependent release dates r_{ik} are respected and $C_{\max} = \max\{C_i | i \in \mathcal{N}\}$ is minimized where C_i is the completion time of job i.

To illustrate the setting of the problem we consider a small example where the underlying structure is a graph.

Example 2.1. *Consider the graph $G = (V, E)$ of Figure 2.1 with $\mathcal{N} = \mathcal{M} = V$. That means that in each node of the graph there is one job and machines can also be located in each node of the graph.*

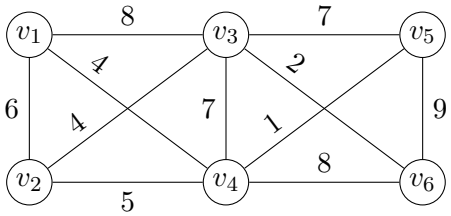

Figure 2.1: Graph $G = (V, A)$ with edge lengths l_{ij} for $v_i, v_j \in V$

Let the number of machines be $p = 2$ and denote with $P = (6, 1, 2, 4, 5, 3)$ the vector of the processing times. The distances $D \in \mathbb{R}^{n \times m}$ are given by shortest paths distances

in the graph, i.e.,

$$D = \begin{pmatrix} 0 & 6 & 8 & 4 & 7 & 12 \\ 6 & 0 & 4 & 5 & 8 & 6 \\ 8 & 4 & 0 & 7 & 7 & 2 \\ 4 & 5 & 7 & 0 & 1 & 8 \\ 7 & 8 & 7 & 1 & 0 & 9 \\ 12 & 6 & 2 & 8 & 9 & 0 \end{pmatrix}.$$

Furthermore, let $\sigma_2 = 1$, $\sigma_4 = 2$, *and* $\sigma_i = 0$ *for* $i = 1, 3, 5, 6$. *Let* $\nu_{41} = 2$, $\nu_{61} = 1/2$, *and* $\nu_{ik} = 1$ *for all other* $i \in \mathcal{N}$ *and* $k \in \mathcal{M}$. *The release dates can be computed from the distances as follows:* $r_{2k} = dist(2, k) + 1$ *for all* k, $r_{41} = \frac{1}{2} dist(4, 1) + 2 = 4$, $r_{4k} = dist(4, k) + 2$ *for* $k \neq 1$, $r_{61} = 2\ dist(6, 1) = 24$, *and* $r_{ik} = dist(i, k)$ *for all other pairs* (i, k). *The matrix of release dates is*

$$R = \begin{pmatrix} 0 & 6 & 8 & 4 & 7 & 12 \\ 7 & 1 & 5 & 6 & 9 & 5 \\ 8 & 4 & 0 & 7 & 7 & 2 \\ 4 & 7 & 9 & 2 & 3 & 10 \\ 7 & 8 & 7 & 1 & 0 & 9 \\ 24 & 6 & 2 & 8 & 9 & 0 \end{pmatrix}.$$

Suppose we want to locate the machines in nodes v_1 *and* v_5. *The jobs have to be assigned to chosen locations and scheduled on the corresponding machine. Since the parallel makespan scheduling problem with release dates is NP-hard in general we cannot do that optimally. In this example let us assign each job to the machine on which it has the smallest release date. This gives the following assignment: jobs 1 and 2 are assigned to* v_1 *and jobs 3,4,5, and 6 are assigned to* v_5. *For the given assignment we can compute an optimal schedule for each machine (by the earliest release date first rule, see Graham [2]) which yields the schedule in Figure 2.2 with a makespan of* $C_{\max} = 14$. *We can easily see that this schedule is not optimal as reassigning job 4 to location* v_1 *yields a makespan of* $C_{\max} = 12$.

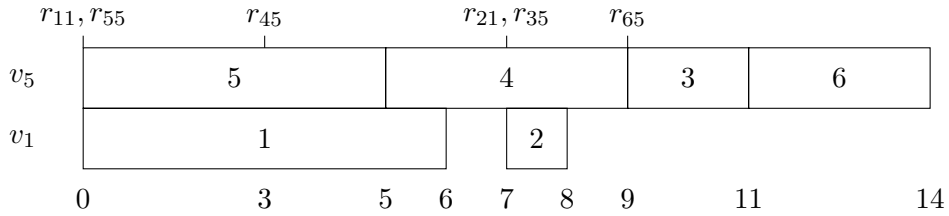

Figure 2.2: Schedule for v_1, v_5 and $A_{v_1} = \{1, 2\}, A_{v_5} = \{3, 4, 5, 6\}$

The example shows that treating the location problem independently of the scheduling problem may yield non-optimal results. Therefore, we present some solution approaches that integrate the scheduling data as good as possible in the location and assignment decisions.

3 IP Formulation and Lower Bounds

Before we consider heuristic solution approaches we formulate the problem as an IP in Section 3.1 to be able to solve at least small scale instances to optimality. Since we cannot solve large scale instances to optimality we need heuristic algorithms. For those instances the gap provided by a commercial IP solver is very high if a feasible solution can be found at all. Therefore, to measure the quality of those algorithms in Section 3.2 we propose several lower bounds for the problem.

3.1 IP Formulation

The DPMM ScheLoc problem combines the two problems of discrete location and parallel machine scheduling. Therefore, we can formulate the problem as an IP by combining the IP formulations of the two separate problems. To do this we introduce scheduling variables x_{ik} for $i \in \mathcal{N}$ and $k \in \mathcal{M}$ which are 1 if job i is processed on a machine in location k and 0 else and scheduling variables y_{ij}^k for $i, j \in \mathcal{N}$ and $k \in \mathcal{M}$ which are 1 if job i directly preceeds job j on machine k and 0 else. Furthermore, we need location variables z_k which are 1 if a machine is placed in location $k \in \mathcal{M}$.

To model the problem as an IP we introduce two dummy jobs 0 and $n+1$ with machine independent release dates equal to 0 and processing times equal to 0. Let M be any upper bound on $C_{\max}$, e.g., $M = \sum_{i \in \mathcal{N}} p_i + \max_{k \in \mathcal{M}} \{r_{ik}\}$. The DPMM ScheLoc Problem can be formulated as IP (3.1)-(3.16).

The objective (3.1) is to minimize the makespan, i.e., the latest completion time of a job. The constraints are split into pure scheduling constraints (3.2) -(3.8) and (3.13) - (3.15), pure location constraints (3.9) and linking constraints (3.10) - (3.12).

Constraints (3.2) - (3.4) compute the completion times and the makespan. The first set of constraints (3.2) make sure that no job is started before its release date, the second set of constraints (3.3) make sure that no job is started before its predecessor is completed. This set of constraints is linearized by the big-M method. To ensure that each job $i \in \mathcal{N}$ has exactly one predecessor and one successor we need constraints (3.5) and (3.6). Here the dummy jobs are used as predecessor of the first job on each machine and as successor of the last job on each machine. Constraints (3.7) make sure that only a job scheduled on machine k can be the first to be processed on that machine and constraints (3.8) ensure that only jobs scheduled on the same machine can be predecessor and successor. The location constraints (3.9) limit the number of locations chosen to p. We have several types of linking constraints: To make sure that only chosen machines are scheduled and each chosen machine is only scheduled once (i.e., not more than one schedule for each machine location) we add constraints (3.10). Constraints (3.11) ensure that jobs are only assigned to opened machines and constraints (3.12) ensure that each job is assigned. The remaining constraints (3.13) to (3.16) are restrictions on the values of variables.

This formulation has a large number of variables ($\mathcal{O}(n^2m)$) and a large number of constraints ($\mathcal{O}(n^2m)$). Therefore, only small problem instances can be solved to optimality using this formulation (see Section 6 for computational results).

$$\min C_{max} \tag{3.1}$$

$$\text{s.t. } C_i \geq \sum_{k\in\mathcal{M}} r_{ik}x_{ik} + p_i, \qquad \forall i \in \mathcal{N} \tag{3.2}$$

$$C_i \geq C_j + p_i - M\left(1 - y_{ji}^k\right), \qquad \forall i,j \in \mathcal{N}, k \in \mathcal{M} \tag{3.3}$$

$$C_{max} \geq C_i, \qquad \forall i \in \mathcal{N} \tag{3.4}$$

$$\sum_{j\in\mathcal{N}\cup\{n+1\}} \sum_{k\in\mathcal{M}} y_{ij}^k = 1, \qquad \forall i \in \mathcal{N}\cup\{0\} \tag{3.5}$$

$$\sum_{i\in\mathcal{N}\cup\{0\}} \sum_{k\in\mathcal{M}} y_{ij}^k = 1, \qquad \forall j \in \mathcal{N}\cup\{n+1\} \tag{3.6}$$

$$y_{0i}^k \leq x_{ik}, \qquad \forall i \in \mathcal{N}, k \in \mathcal{M} \tag{3.7}$$

$$y_{ij}^k \leq \frac{1}{2}\left(x_{ik} + x_{jk}\right), \qquad \forall i,j \in \mathcal{N}, k \in \mathcal{M} \tag{3.8}$$

$$\sum_{k\in\mathcal{M}} z_k \leq p \tag{3.9}$$

$$\sum_{j\in\mathcal{N}} y_{0j}^k \leq z_k, \qquad \forall k \in \mathcal{M} \tag{3.10}$$

$$\sum_{j\in\mathcal{N}} x_{jk} \leq nz_k, \qquad \forall k \in \mathcal{M} \tag{3.11}$$

$$\sum_{k\in\mathcal{M}} x_{ik} = 1, \qquad \forall i \in \mathcal{N} \tag{3.12}$$

$$C_0 = 0 \tag{3.13}$$

$$C_i \geq 0, \qquad \forall i \in \mathcal{N} \tag{3.14}$$

$$y_{ii}^k = 0, \qquad \forall i \in \mathcal{N}, k \in \mathcal{M} \tag{3.15}$$

$$x_{ik}, y_{ij}^k, z_k \in \{0,1\}, \qquad \forall i,j \in \mathcal{N}, k \in \mathcal{M}. \tag{3.16}$$

We can improve the formulation by relaxing the integrality constraints on variables x_{ik} and z_k. This does not change the formulation which can be seen as follows: Suppose there is some $x_{ik} \in (0,1)$. If job i is scheduled first on machine k then $y_{0i}^k = 1$ since y_{ij}^k are still binary variables. So by constraints (3.7) also $x_{ik} = 1$. Inductively assume that x_{lk} integral for the first l jobs on machine k. Then for the $(l+1)$st job on machine k we get $x_{l+1,k} = 1$ by constraints (3.8) together with the assumption that $y_{l,l+1}^k = 1$ and $x_{lk} = 1$. Therefore, all x_{ik} are integral without enforcing it by constraint (3.16). Next

suppose that some z_k is non-integral. If there is at least one job scheduled on machine k we get that $z_k = 1$ by constraints (3.10). If no job is scheduled on machine k the variable z_k may be non-integral, but by constraint (3.9) we get that there are at most p locations for which $z_k = 1$ and jobs are scheduled only in those locations. To obtain an optimal integral solution we set $z_k = 0$ for all non-integral z_k which is feasible since there are no jobs scheduled on the corresponding machines.

Unfortunately, this relaxation does not improve the computation time to solve the IP since we still require y_{ij}^k to be integral. If we relax those variables, constraints (3.3) will always result in $C_i \geq C_j + p_i - M'$ for some large value M' which makes the constraints superfluous. In an optimal solution of the relaxation the completion times will only be determined by constraints (3.2) yielding a very bad lower bound.

Since the IP formulation can only be solved to optimality for small scale instances we need to find heuristic algorithms to solve the problem. Furthermore, to measure the quality of the heuristics we need lower bounds since we cannot compare the heuristic values to optimal solutions. Therefore, in the next subsection we introduce various lower bounds for the problem.

3.2 Lower Bounds

To measure the quality of the heuristics we present several lower bounds for the DPMM ScheLoc Problem. Probably the easiest lower bound can be found by considering the pure scheduling problem $P_m||C_{\max}$, i.e., the parallel machine scheduling problem that minimizes the makespan without release dates. This is a special case of the DPMM ScheLoc Problem where $D = \mathbf{0}$, $\sigma_i = 0$ for all $i \in \mathcal{N}$ and $\nu_{ik} = 1$ for all $i \in \mathcal{N}$ and $k \in \mathcal{M}$. For this problem a well-known lower bound is the following:

$$C_{\max} \geq \frac{\sum_{i \in \mathcal{N}} p_i}{p}. \tag{3.17}$$

Since $r_{ik} = 0$ for all $i \in \mathcal{N}$ and $k \in \mathcal{M}$ is a lower bound on the release dates this also is a lower bound for the DPMM ScheLoc Problem. Of course the bound will be very weak for problems with many $r_{ik} > 0$. However, if there are at least p locations k for which $\min_{i \in \mathcal{N}} r_{ik} = 0$ we can show that the bound is tight.

Example 3.1. *Let the graph $G = (V, A)$ in Figure 3.1 be given. Let $\mathcal{N} = \mathcal{M} = V$, i.e., there is one job in each node and each node is a possible machine location. We want to locate $p = 2$ machines. The processing times are given by $P = (2, 3, 5, 3, 5, 2)$.*

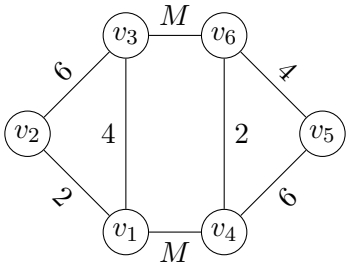

Figure 3.1: Graph $G = (V, A)$ with $dist_{ij}$ for $v_i, v_j \in V$

If we locate the machines in locations v_1 and v_6 we get as optimal schedule the sequence (1,2,3) on machine 1 and (6,4,5) on machine 6. The makespan is $C_{\max} = 10$ which is equal to $\frac{\sum p_i}{p} = \frac{20}{2}$.

Note that for the tightness of the bound it is only important that $r_{11} = 0$ and $r_{66} = 0$. All other release dates may be unequal to zero in a randomly generated setting. The example can easily be extended to work for arbitrary values of n and p.

The tightness of the bound implies that we can only enhance the bound by a part that is equal to zero if the above property is fulfilled. One way to do that is to add the minimum release date:

$$C_{\max} \geq \frac{\sum_{i \in \mathcal{N}} p_i}{p} + \min_{k \in \mathcal{M}} \min_{n \in \mathcal{N}} r_{ik}.$$

When job locations are allowed as machine locations this bound is equivalent to the previous bound since the minimum release date will be zero.

To improve the bound in case that all job locations are possible machine locations (i.e., $\mathcal{N} = \mathcal{M}$) we consider $r_i = \min_{k \in \mathcal{M}: k \neq i}\{r_{ik}\}$ the smallest release date of job i if it is not processed in its job location. If there is no machine located in i a lower bound on the completion time of job i is $LB(C_i) = r_i + p_i$. Otherwise, $LB(C_i) = p_i$. A lower bound on the makespan is $C_{\max} \geq \max\{LB(C_i)\}$. Therefore, we assume that the machines are located in the p locations i for which $r_i + p_i$ is maximal. Let K' be the set of the corresponding jobs. For all other jobs we get $LB(C_i) = r_i + p_i$ and can bound the makespan by

$$C_{\max} \geq \max_{i \notin K'}\{r_i + p_i\}.$$

Finally, we propose a bound that not only considers the minimum release date but also the structure of the release dates. For each $k \in \mathcal{M}$ we optimally solve a single machine problem with respect to all jobs but only until the load $l(k)$ of machine k is $l(k) = \frac{\sum_{i \in \mathcal{N}} p_i}{p}$. If this load is achieved during the processing of a job, this job is only partially processed and the makespan of machine k is the completion time of this partial job. The optimal scheduling can be done by the earliest release date (ERD) rule: Whenever a machine is free, schedule the available job with smallest release date. Let $C_{\max}(k)$ be the makespan on machine k of this subset of jobs. We get the following lower bound:

$$C_{\max} \geq \min_{k \in \mathcal{M}}\{C_{\max}(k)\}. \tag{3.18}$$

Correctness of the bound can be seen as follows: In any feasible solution of the ScheLoc Problem at least one machine has load $l(k) \geq \frac{\sum_{i \in \mathcal{N}} p_i}{p}$ since otherwise jobs remain unscheduled. Since the ERD rule is optimal for the single machine problem the point in time $C_{\max}(k)$ is the minimal completion time of machine k for $l(k) \geq \frac{\sum_{i \in \mathcal{N}} p_i}{p}$. Let location k' be such that $k' \in \arg\min_{k \in \mathcal{M}}\{C_{\max}(k)\}$. If location k' is chosen in an optimal solution and has $l(k') \geq \frac{\sum_{i \in \mathcal{N}} p_i}{p}$ the lower bound holds. Otherwise, there is some other machine k'' that fulfills $l(k'') \geq \frac{\sum_{i \in \mathcal{N}} p_i}{p}$ with $C_{\max}(k'') \geq C_{\max}(k')$. Therefore, LB (3.18) indeed is a lower bound.

4 Clustering Heuristics

The DPMM ScheLoc Problem is an NP-hard problem. Since we can only solve small scale instances using the IP formulation and commercial IP solvers we want to find heuristic algorithms that provide good solutions in a short computation time. The problem decomposes into three different problems: Locating the machines, assigning the jobs to the machines, and scheduling the assigned jobs on each machine. Once the machine locations have been chosen and the jobs have been assigned to the machines, the remaining scheduling problem can be solved optimally by the ERD rule. Therefore, we want to find heuristics that solve the first two subproblems. To obtain good results we want to integrate the scheduling data in the location and assignment decisions. An approach that allows to solve each subproblem iteratively while considering data of the other subproblems is to solve clustering problems. Formally, the clustering problem is defined as follows: Given a discrete set of possible locations $\mathcal{M}$, a set of demand locations $\mathcal{N}$ with demand d_i for each $i \in N$ and a distance function $dist\,(i,k)$ for $i \in \mathcal{N}$ and $k \in \mathcal{M}$, find a subset $\mathcal{C} \subset \mathcal{M}$ of size p called cluster centers and an assignment $C_k \subset \mathcal{N}$ of demand locations to cluster centers $k \in \mathcal{C}$ such that $\bigcup_{k \in \mathcal{C}} C_k = \mathcal{N}$, $C_k \cap C_l = \emptyset$ for all $k, l \in \mathcal{C}$, and $\sum_{k \in \mathcal{C}} \sum_{i \in C_k} d_i dist\,(i,k)$ is minimized. In case of uncapacitated clusters, each demand location i is assigned to the closest cluster center C_k independent of the values d_i.

The DPMM ScheLoc Problem can be formulated as a clustering problem by identifying the cluster centers C_k with the machine locations k and the demand locations with the job locations. Finding cluster centers and an assignment of demand locations to cluster centers corresponds to choosing machine locations and assigning jobs to machines. However, in the DPMM ScheLoc Problem we want to optimize a different objective function. Therefore, we alter the assignment criterion such that the scheduling data is taken into account.

We considered three types of heuristics to solve the clustering problem:

CH1 First choose the cluster centers then assign the jobs to the clusters.

CH2 First cluster the jobs in p clusters then assign a cluster center to each cluster.

CH3 Iteratively choose a cluster center and the jobs assigned to that cluster center until there are p clusters.

For each of the three types we identified various criteria based on which the location and the clustering decisions are made. In the following subsections we present the three types of clustering heuristics and the criteria in more detail.

4.1 Location first, Cluster second

In this type of clustering heuristic we first compute all cluster centers and then assign the jobs to the clusters. We identified several criteria to choose the cluster centers.

CH1-1 Select positions arbitrarily.

CH1-2 Select machine positions k such that $\max_{i \in \mathcal{N}} r_{ik} + p_i$ is minimal.

CH1-3 Select machine positions k such that $\sum_{i \in \mathcal{N}} r_{ik}$ is minimal.

CH1-4 Select machine position k such that r_{i^*k} with $i^* \in \arg\max_{i \in J} \left\{ \sum_{l=1}^{k-1} r_{il} \right\}$ is minimal.

The easiest criterion is to choose the locations arbitrarily and integrate the scheduling data only in the assignment decision. A better way is to already integrate the scheduling data in the location decisions as is done in the other three criteria. In the second criterion we compute a lower bound on the completion time for each job on each machine, namely the sum of release date and processing time. We choose the locations such that the maximum of the lower bounds is minimal. This criterion selects machine locations that are close to the jobs with long processing times avoiding that a large job arrives late at a machine which increases the makespan if the machine has been idle before. The problem with this criterion is that machines may be far away from small jobs which increases the makespan if a lot of small jobs arrive at the machine at the same late point in time. Therefore, in criterion three we take the sum over all release dates and select the machines that minimize that sum. This criterion selects machines that are not too far away from all the jobs such that many jobs will arrive early at the machine and the idle time will be small. The problem with criteria one to three is that all machines are selected at the same time. An even better way is to choose locations one by one and take previous decisions into account. This is done in criterion four. We select the first machine position randomly. In iterations $k = 2, \ldots, p$ the first $k-1$ machine locations are already selected. We compute the sum of the release dates for each job i on the already selected machines $1, \ldots, k-1$. Location k is then chosen such that it is closest to the job i^* for which the sum of the release dates is maximum. That means, the new machine location is selected such that it is close to the job that is far away from all the previously selected machines. This criterion ensures a better spread of the machines. However, it may select locations that are close to one job but far away from all others. This problem is overcome in the second and third version of the clustering heuristic (see Subsections 4.2 and 4.3).

To compute the assignment we use a modified ERD rule: Whenever a machine is free schedule the available job with smallest release date on that machine next. This rule has

two advantages: It well integrates the scheduling data, both the release dates and the processing times and it automatically computes the optimal schedule for the resulting assignment as on each machine the jobs are ordered according to non-decreasing release dates. The procedure is summarized in Algorithm 1.

Algorithm 1 Location first, Cluster second

Input: Instance of a DPMM ScheLoc Problem

Output: Set of p machine locations and a schedule for each machine

1: Compute the locations using one of the criteria 1-4.
2: Schedule the jobs by the modified ERD-rule.

4.2 Cluster First, Location Second

In this version of clustering heuristic we first compute the clusters and then assign a location to each of the clusters. Since the machine locations, and therefore the release dates are unknown when the clustering is done, we can only do that based on the distances between job locations. As these distances are not part of the original input of the problem we cannot compute this version of clustering heuristic for all instances of the problem. But if the underlying structure of the problem is a network (see Examples 2.1 and 3.1) or a plane (job and machine locations are given by coordinates in $\mathbb{R}^2$) we can compute the distances between jobs as shortest paths or by the Euclidean distance.

To do the clustering we first have to select a starting job for each cluster. This can be done by various criteria:

CH2-1 Choose p random jobs.

CH2-2 Let G be the center of gravity of all jobs. Select the p jobs i as cluster centers that maximize $dist(G, i)$.

CH2-3 Let G_i be the center of gravity of selected cluster centers $1, \ldots, i$. Select cluster center $i + 1$ such that it maximizes $dist(G_i, i + 1)$.

The first criterion randomly chooses the cluster centers. The second criterion chooses all cluster centers at the same time based on the distance to the center of gravity. This criterion puts jobs in different clusters that are far from the center of the jobs. This ensures some scattering but also allows jobs that are close to each other to be in different clusters if they are both far from the center of gravity. To overcome this problem we again propose an iterative procedure that given i cluster centers chooses the $i + 1$st cluster center such that it is furthest away from the center of gravity of the i cluster centers. This criterion avoids putting jobs in different clusters that are far from G but close to one another. Given the cluster centers we place the remaining jobs j one by one in the cluster that has its center of gravity closest to job j.

The advantage of the cluster first, location second heuristic is that for each computed cluster we can find the optimal location as we can compute the optimal schedule for each

location. But if more than one cluster has the same optimal location to avoid trying each possible combination of p locations we apply an iterative heuristic in this case: For each cluster of jobs, compute the makespan at each location and order the locations by non-decreasing makespan. If there is a location k that is optimal for more than one cluster find the cluster that has the worst second-largest makespan. Assign this cluster to location k and move all other clusters to the respective locations where the clusters have the second-largest makespan. Continue this procedure comparing the next-largest makespan of each cluster competing for the same location (e.g., if cluster i is currently at the location with the l-largest makespan and cluster j at the location with the r-largest makespan compare the $l+1$st-largest makespan of cluster i with the $(r+1)$st-largest makespan of cluster j) until all clusters are assigned uniquely to a location. For $p \leq m$, i.e., less clusters than possible locations this procedure will terminate after at most p^2 steps (this bound is reached in case all p clusters have to be moved to the p-worst location). This is due to the fact that a location that is assigned to a cluster in some step of the algorithm will always be assigned to a cluster in all following steps as clusters are only moved if there is more than one cluster at the same location.

The second type of clustering heuristic is summarized in Algorithm 2.

Algorithm 2 Cluster First Location Second

Input: Instance of a DPMM ScheLoc Problem

Output: Set of p machine locations and a schedule for each machine

1: Determine the p cluster centers using one of the criteria 1-3.
2: For each remaining job j put it into the cluster with minimal distance between its center of gravity and j.
3: For each cluster compute the makespan for each location and assign each cluster to the location with smallest makespan.
4: If more than one cluster is assigned to the same location:
Schedule the cluster with worst next largest makespan at this location. Move all other clusters to the location with next largest makespan.
5: Repeat Step 4 until every cluster is assigned uniquely to a location.

4.3 Iterative Selection of Clusters and Locations

In the first two versions of the clustering heuristic introduced in Subsections 4.1 and 4.2 we solve the location and assignment problem sequentially. Instead we can consider a third type of heuristic where we iterate between the location and the assignment problem. The advantage of this approach is that we can use the partial results of both subproblems in the course of the algorithm.

In this heuristic we start with a single cluster that contains all jobs. For this cluster we can compute a single optimal machine location. In the next step we compute a set of jobs to be removed from the existing cluster and compute a new optimal location for this cluster. We continue this procedure until we reach the maximum number p of locations. There are different possibilities to choose the locations and the assignment. For the location part we identified the following criteria:

1. Select as next machine location the one that minimizes the makespan of all unassigned jobs.
2. Select as next machine location the one that minimizes the makespan of the $\lfloor n/p \rfloor$ jobs with smallest release date on that machine.

The first criterion selects a machine location that is fit for all the unassigned jobs while the second criterion only minimizes the makespan of the $\lfloor n/p \rfloor$ jobs that are closest to the machine. This criterion assumes that on each machine the same number of jobs is scheduled. This approach is useful if the processing times of the jobs are similar. Otherwise, the first criterion is the better selection as it does not make any assumptions on the assignment that is not yet computed but tries to find a location that is good for all remaining jobs.

For the assignment part we propose the following criteria:

1. Consider the optimal schedule of all unassigned jobs on the selected machine k. Find the first job j with $C_j > r_{jk} + p_j$, remove this job from the machine, and compute a new optimal schedule (by moving all jobs i with $C_i > C_j$ as far to the left as possible). Continue until there are no more jobs with $C_j > r_{jk} + p_j$.
2. Assign the $\lfloor n/p \rfloor$ jobs with smallest release dates on k to the machine.

The first criterion removes all jobs from the cluster that cannot start at their release date. Its aim is to find a schedule in which most jobs have the smallest possible completion time on the machine to which they are assigned. But if the release dates are not well distributed this may lead to a schedule where one machine has many jobs while others have only few jobs. This is the case as we do not know how many jobs are removed from the cluster. If that are only very few, the last clusters will be empty, if that are many the last cluster will be overfull. Therefore, we introduce the second criterion which creates clusters of the same size with respect to number of jobs. Since the second location criterion chooses an optimal location for an assignment that coincides with the second assignment criterion we combine this location criterion only with the second assignment criterion and obtain the following three heuristics:

CH3-1 Location criterion 1, assignment criterion 1

CH3-2 Location criterion 1, assignment criterion 2

CH3-3 Location criterion 2, assignment criterion 2

The heuristic is summarized in Algorithm 3.

Algorithm 3 Clustering Heuristic 3

Input: Instance of a DPM ScheLoc Problem

Output: Set of p machine locations and a schedule for each machine

1: Select a machine location k using location criterion 1 or 2.
2: If less than p locations are selected, assign jobs to location k using assignment criterion 1 or 2. Go to Step 1.
3: If p locations are selected, assign all remaining jobs to location k.

The main problem of this heuristic is that all remaining jobs are assigned to the last selected machine. Especially with assignment criterion 1 this can lead to an unbalance in the load of the machines. To overcome this problem we can reassign some of the jobs at the end of the clustering heuristic. We call this post-optimization procedure and present details in Subsection 4.4.

4.4 Post-Optimization Procedure

As mentioned in the previous section the load on the machines may be unbalanced after the iterative clustering heuristic is performed. This may also be the case for the other two types of clustering heuristics. Therefore, we introduce a post-optimization procedure that tries to balance the load on the machines by reassigning some jobs. The procedure starts with a feasible solution to the DPMM ScheLoc Problem and selects a machine k that currently determines the makespan. It tries to reassign the last job j from k to any other selected machine. This can be done by putting j in its optimal position (according to the ERD-rule) on each machine. If for some machine l this yields an overall improvement of the makespan, we move job j to machine l and iterate. If there is no machine such that the makespan is improved the procedure stops. Details are presented in Algorithm 4.

Algorithm 4 Post-Optimization Procedure

Input: Instance of a DPMM ScheLoc Problem, feasible solution
Output: Improved feasible solution

1: Determine machine k that currently determines the makespan.
2: Select the job j that is currently scheduled last on machine k.
3: For $l = 1, \ldots, p$ the set of selected machines, schedule job j on $l \neq k$ in the position of the ERD-order.
4: If there is a machine $l \neq k$ such that the makespan of the solution is improved if job j is moved to machine l:
Remove job j from machine k and put it in its ERD-position on machine l.
Go to Step 1
5: Else, stop. Output the (new) solution.

Note that the new solution is indeed at least as good as the old one since we only perform changes that yield a better makespan.

5 Local Search

A major drawback of the clustering heuristics is that locations and assignments are fixed when they are chosen and cannot be altered in later iterations. Even with the post-optimization procedure the possibilities to alter bad decisions are very limited. To overcome this problem we propose a local search heuristic that iterates between choosing locations and clusters until a local optimum is reached. The iterations are done by combining the location first and the cluster first heuristics of Subsections 4.1 and 4.2.

The local search starts with a feasible solution of the DPMM ScheLoc Problem, e.g., the result of one of the clustering heuristics. For the selected locations of this solution we compute a new assignment of jobs to machines by the modified ERD-rule as in the location first heuristic. If this new set of clusters improves the makespan compared to the starting solution, we continue with the new clusters, otherwise we continue with the clusters of the current solution. In the next step a new set of locations for the clusters is computed as in the cluster first heuristic. If the new solution is better than the old one, we restart the procedure for the new set of locations. Otherwise, the search stops and outputs the current solution that is at least as good as the input solution. The search is summarized in Algorithm 5.

Algorithm 5 Local Search

Input: Instance of a DPMM ScheLoc Problem, feasible solution $S = (X, A)$ with selected locations X and selected assignment A.

Output: Improved feasible solution

1: For the locations X compute an assignment A' by the modified ERD-rule.
2: If A' improves the makespan set $A = A'$.
3: For assignment A compute new locations X' by the ranking procedure of Section 4.2.
4: If X' improves the makespan set $X = X'$. Else Stop and output $S = (X, A)$.

To guarantee a polynomial runtime the number of iterations has to be bounded as the local optimum is not necessarily found in a polynomial number of iterations. However, computational tests showed that in practice the search terminates in very short computation time (see Section 6).

6 Computational Results

To test all proposed heuristics as well as the proposed IP formulation we performed extensive computational tests. To this end we created four different instance sets. The first two sets consist of random instances, i.e., instances without any structure in the distances. The first set contains 50 small scale instances with up to 30 jobs, 10 locations and 8 machines. The second set contains 450 larger instances with up to 300 jobs, 60 locations and 50 machines. For the third set 350 networks were randomly generated. The ScheLoc instances were obtained by setting $V = \mathcal{N} = \mathcal{M}$ and computing the distances as lengths of shortest paths in the graph. The number of machines was randomly generated. Instances in this set have up to 300 jobs (i.e., up to 300 possible machine locations) and up to 35 machines. For the last set 600 instances with up to 300 jobs and 35 machines were generated as follows: jobs were randomly located in squares of different size in the plane and identified with the set of possible machine locations, i.e., $\mathcal{M} = \mathcal{N}$. Distances were computed as euclidean distance and the number of machines was randomly generated.

For all instances each heuristic with and without the post-optimization procedure was tested. For the randomized versions of the heuristics 10 runs where performed on each

instance and best and average values stored. To test the local search the best solution of each of the 10 clustering heuristics with post-optimization were used as starting solutions. Furthermore, 10 runs with random starting solution were performed. For small scale instances the results of the heuristics were compared to the results of the IP formulation using the commercial IP solver Cplex and a time bound of 15 minutes. Results of this test can be found in Subsection 6.1. For the larger instances the results were compared to the lower bounds proposed in Subsection 3.2. Results of this test can be found in Subsection 6.2.

All tests were performed on a 16-core Intel Xeon E5-2670 processor, running at 2.60 GHz with 20MB cache, 96 GB RAM and Ubuntu 12.04. The IP tests were done using multi-threading with five threads.

6.1 Comparison to IP solver

For the 50 instances of testset 1 as well as for 25 small scale instances of testset 3, and 50 small scale instances of testset 4 with up to 20 jobs, 20 locations and 5 machines the IP was solved by the commercial IP solver Cplex. A time bound of 15 minutes per instance was enforced. Details on the IP tests can be found in Table 6.1. Cplex was run on testset 1 also for 60 minutes but there was no improvement on the best found solution or the number of optimal solutions found.

n / m / p	N° Instances	Avg Gap	N° Optimal	Avg time (OPT)	Avg time (Best found)
Testset 1	50	28.6%	26	35.9s	53.0s
10 / 10 / 8	20	0.0%	20	23.8s	1.2s
30 / 10 / 9	30	52.0%	6	76.1s	90.0s
Testset 3					
20 / 20 / 5	25	47.6%	2	271.85s	194.0s
Testset 4					
20 / 20 / 5	50	51.3%	2	466.7s	170.8s

Table 6.1: Results of Cplex for runtime bound of 15 minutes

The table shows the number of instances per testset, the average gap reached after 15 minutes of computation time, the number of instances solved to optimality, the average time needed to solve those instances to optimality and the average time over all instances needed to find the best solution without proving optimality. The 50 instances of testset 1 are split into 20 instances with up to 10 jobs and 30 instances with 11-30 jobs. Each value is denoted for the entire set as well as for the two subsets.
Testset 1 contains 20 instances with at most 10 jobs. For all of them Cplex was able to find an optimal solution. For the 30 instances with more than 10 jobs Cplex was only able to find the optimal solution for 6 instances with up to 15 jobs and 5 locations. The average time for the 26 optimally solved instances to find and proof optimality is 35.9 seconds. The average time over all instances to find the best known solution is 53 seconds but for the 30 instances with more than 10 jobs the average time is increased

to 90 seconds. The average gap is 28% which is due to the large number of optimally solved instances. For the 30 instances with more than 10 jobs the gap increases to 52%. This is most likely due to the bad lower bounds computed by Cplex.

For testset 3 Cplex was tested on 25 instances with 10-20 jobs. The optimal solution could only be found for two instances in on average 271 seconds. The average time to find the best known solution is 194 seconds. The increased computation time compared to the first testset (even if only instances with 10 or more jobs are considered) is due to the larger number of possible machine locations that for testset 3 is identical to the number of jobs (at least 10) while for testset 1 it is at most 10. This also explains why fewer optimal solutions are found. The average gap of 47.6% is comparable to that of testset 1 if only instances with 10 or more jobs are considered.
For testset 4 Cplex was tested on 50 instances with 10-20 jobs. The optimal solution could only be found for two instances in on average 466 seconds. The average time to find the best known solution is 171 seconds. The values are comparable to those of testset 3 as the instances are of the same size in terms of number of possible locations and number of jobs. There is no difference in the performance although the instances of the testsets have a different structure. The average gap of 51.3% is comparable to that of testset 2 and of testset 1 for the instances with 10 or more jobs.

Since Cplex very often does not find the optimal solution we compare the results of Cplex to those of the clustering heuristics. For better readability we only use the best results of each version of the clustering heuristics (CH1: location first, CH2: cluster first, CH3: iterative). The same holds for the local search, we only use the best results of the local search for starting solutions from each version of the clustering heuristics and of the ten runs with random starting solutions (LS1: CH1 as starting solution, LS2: CH2 as starting solution, LS3: CH3 as starting solution, LS Rand: Random starting solution). A detailed comparison of different criteria for the same type of clustering heuristic can be found in Subsection 6.2. Table 6.2 shows the percentage of deviation from the best known solution which in all but four cases is the solution of Cplex. As explained in Subsection 4.2 the cluster first heuristic is not tested on randomly generated instances but only on network and planar instances. Since the runtime of each clustering heuristic and the local search is only in the order of milliseconds we only compare the quality of the solutions.

	CH1	CH2	CH3	LS1	LS2	LS3	LS Rand
Testset 1	11.7	-	17.5	5.5	-	5.7	5.1
Testset 3	13.4	21.1	17.1	8.2	9.4	7.4	3.4
Testset 4	7.5	19.5	15.8	6.6	8.7	8.7	3.0

Table 6.2: Average Deviation (in percent) of clustering heuristics from best known solution

Table 6.2 shows that all clustering heuristics provide results that are on average 7-21% from the best known solution. If the local search is additionally applied the gap is reduced to 3-9% on average. We also see that the best performance is obtained by repeatedly starting the local search with a random starting solution. This is due to the

ten runs that are performed with random starting solutions that cover many of the for small scale instances rather limited location choices. If instead for each instance the best solution over all local searches with starting solutions of the clustering heuristics is taken, the gap can be further reduced to 2-3%. Since each run of the heuristics and the search can be done in a few milliseconds this approach in total will still be very fast (around 5-10 seconds). Therefore, the heuristic approach is competitive with Cplex as it drastically improves the runtime at only a very small cost in terms of solution quality.

6.2 Comparison of Heuristics

For almost all large scale instances Cplex was not able to even find a feasible solution within the 15 minutes time bound. Therefore, in this section we compare the results of the different heuristics on testsets 2-4 to the best one of the lower bounds introduced in Subsection 3.2.

6.2.1 Post-Optimization Procedure

First we want to analyze the improvement that the post-optimization procedure introduced in Subsection 4.4 yields. We tested all heuristics with and without the post-optimization procedure. This procedure is implemented such that the heuristics with post-optimization always perform at least as good as without post-optimization. But it is not a priori clear whether the post-optimization yields a significant improvement. The results presented in Table 6.3 however show that this indeed is the case. For the randomized heuristics again ten runs were performed and the best solution is denoted. The results with post-optimization shown in Table 6.4 are obtained by performing the post-optimization on the solutions without post-optimization, i.e., the randomization has no influence on the comparability of the values.

	CH1-1	CH1-2	CH1-3	CH1-4
Set 1	18.6	19.8	19.7	19.0
Set 3	17.8	22.7	21.7	20.5
Set 4	22.9	27.3	24.5	25.2

	CH2-1	CH2-2	CH2-3	CH3-1	CH3-2	CH3-3
Set 1	-	-	-	735.6	175.0	173.7
Set 3	42.7	56.5	64.3	715.5	104.6	120.9
Set 4	40.5	50.8	70.4	783.0	116.8	128.0

Table 6.3: Average Deviation (in percent) of clustering heuristics without post-optimization from best lower bound

	CH1-1	CH1-2	CH1-3	CH1-4
Testset 2	17.9	18.2	18.4	17.8
Testset 3	16.0	21.2	20.8	19.4
Testset 4	19.6	24.5	22.8	22.4

	CH2-1	CH2-2	CH2-3	CH3-1	CH3-2	CH3-3	Best CH
Set 2	-	-	-	22.9	24.3	25.8	12.3
Set 3	18.1	25.2	30.1	21.2	20.2	24.0	9.5
Set 4	19.2	23.4	34.5	17.6	18.4	21.1	11.5

Table 6.4: Average Deviation (in percent) of clustering heuristics with post-optimization from best lower bound

From the values we can see that especially for the cluster first and the iterative heuristics there is a major improvement by the post-optimization. For the cluster first heuristic that is due to the fact that clustering is done independent of processing times. For CH3-1 it is due to the misbalance in number of jobs assigned to the different machines and for the other two iterative heuristics it is due to the misbalance in processing times as the number of jobs is equalized. This shows that for these two types of heuristics the post-optimization procedure is essential to improve the assignment. But since the solution with post-optimization is good that shows that these heuristics make good location choices. For the location first heuristic the improvement is only around 1-3% but since there is no notable increase in the runtime of the heuristics it is worth including the procedure in all of the heuristics. Therefore, in the following we only compare the results of the heuristics with post-optimization.

6.2.2 Comparison to Lower Bounds

Table 6.4 shows the average deviation from the lower bound for all 10 versions of the clustering heuristics with post-optimization using the best solution for the randomized heuristics, where the numbering is according to that given in Subsections 4.1-4.3. The last column Best CH gives the average deviation if for each instance the best solution over all clustering heuristics is taken. As explained in Subsection 4.2, CH2 is not tested for instances with random distances.

As can be seen in the table the different criteria of the same version yield comparable average results. There are slight differences in the versions of the heuristics but there is no version that performs best on all testsets. While CH1 performs best on the random and network instances CH3 performs best on planar instances. The results of CH2 are of comparable quality but are slightly worse for both network and planar instances. This may be due to the fact that in this heuristic the clustering is done without considering processing times. Altogether the randomized heuristics (CH1-1, CH1-4 and CH2-1) perform best among their respective versions. But this only holds for the best solution over the ten runs. Table 6.5 shows that for the random clustering heuristics the average value is around 1-6% worse than the best value of the same heuristic. The best results can be achieved if for each instance all ten versions of the heuristics are run and the

best result is taken (CH Best) which improves the gap by 5-8%. This shows that the performance of each heuristic is dependent on the problem data but for each instance there is at least one version of the heuristic that performs quite well. This approach is valid since the runtime of each single heuristic is within milliseconds yielding a runtime of only a few seconds to obtain the Best CH solution.

	CH1-1 avg	CH1-1 best	CH1-4 avg	CH1-4 best	CH2-1 avg	CH2-1 best	LS Rand avg	LS Rand best
Set 1	18.4	17.9	18.1	17.8	-	-	29.1	17.7
Set 3	21.6	16.0	23.3	19.4	22.8	18.1	15.9	9.7
Set 4	25.2	19.6	27.3	22.4	23.3	19.2	16.0	10.0

Table 6.5: Average Deviation (in percent) of randomized clustering heuristics (best and average solution value) from best lower bound

Table 6.5 shows that for CH1-1 the average values are comparable with the values of CH1-2 which is the worst heuristic of the location first type. The average values of CH1-4 are even worse and around 4% from those of CH1-2 for the network and planar instances. For CH2-1 the average values are around 2% better than those of CH2-2 and even 7-11% better than those of CH2-3. That shows that a random selection of cluster centers performs quite well compared to a selection of cluster centers based on centers of gravity. For the local search with random starting solution the average values are even worse with around 6-12% deviation from the best run. For testset 1 with the random distances the average value is even around 10% worse than the average value of the random clustering heuristics (CH1-1, CH1-4, CH2-1). For the network and planar instances on the other hand the average value of the local search with random starting solution is around 3% better than the best solution of the clustering heuristics. This shows that the local search performs quite well even with an arbitrary solution if the instance has a special structure. However, in Section 6.2.3 we will see that if the local search is applied to the solutions of the clustering heuristics the average value of the local search with random starting solution is comparable to the worst solutions and around 3% worse than the best solution (see Table 6.6).

6.2.3 Local Search

To analyze the improvement the local search yields on different starting solutions, Table 6.6 shows the average deviation of the local search with all different starting solutions tested. For randomized heuristics the local search is always started with the best solution of the ten runs. For the local search with random starting solution the best solution of the ten runs is denoted. The last column LS Best denotes the deviation if for each instance the best solution over all local searches with clustering starting solutions (i.e., not with random starting solution) is taken.

	CH1-1	CH1-2	CH1-3	CH1-4	CH2-1	CH2-2	CH2-3
Testset 1	14.1	14.5	14.6	14.2	-	-	-
Testset 3	13.0	14.8	15.7	14.3	13.8	16.9	17.0
Testset 4	13.7	15.4	16.2	12.8	14.1	15.3	16.3

	CH3-1	CH3-2	CH3-3	Rand	Best
Testset 1	14.3	15.3	15.5	17.7	10.7
Testset 3	15.1	13.9	16.1	9.7	8.2
Testset 4	13.5	13.8	14.8	10.0	8.9

Table 6.6: Average Deviation (in percent) of local search from best lower bound

Comparing the results to those of Table 6.4 shows that the local search yields a significant improvement for each clustering heuristic of at least 4%. In the best cases this improvement is even 18%. It can also be seen that in general the better the starting solution the better the outcome of the local search. There are only few exceptions in the range of only 1%, e.g., for testset 3 the local search with CH3-2 is slightly better than that of CH1-4 (0.4%) while the solution of CH3-2 is slightly worse than that of CH1-4 (0.8%). For testsets 3 and 4 the local search with random starting solution performs best among each single heuristic. That is due to the ten runs performed for that heuristic which allow to cover a larger part of the search space than with any single starting solution of the clustering heuristics. As mentioned above the average values are comparable with the worst solutions starting with clustering solutions. If instead for each instance the best solution over all local searches starting with a clustering solution is taken, we obtain a solution that is 1-2% better than that of the best random starting solution for network and planar instances and even 7% better for the random instances. Since each heuristic runs in a few milliseconds we can compute the values of column Best in around 5-10 seconds.

7 Conclusion and Further Research

We considered an integrated Scheduling-Location Problem, the Discrete Parallel Machine (DPMM) ScheLoc Problem. To our knowledge this is the first work done on algorithms for Parallel Machine ScheLoc Problems. We gave an IP formulation which unfortunately can only be solved by commercial IP solvers for small scale instances. Therefore, we give several clustering type heuristics together with a post-optimization procedure, as well as a local search heuristic. Tests showed that all clustering heuristics with post-optimization performed quite good and are 16-35% from a lower bound. All heuristics run within milliseconds. Therefore, it is a feasible approach to run all clustering heuristics and take the best solution found. This procedure reduces the gap to 10-12%. If additionally the local search is used with the clustering solutions as starting solutions this procedure further reduces the gap to around 8-10%.

These results show that the proposed heuristics are a first good approach to the problem. However, more elaborate search heuristics like a variable neighbourhood search

may be able to further improve the results. Also efficient exact solution methods like a branch-and-bound approach are worth investigating.

In general there is very little research on ScheLoc Problem although it is an interesting combination of two well-studied optimization problems with many applications. Therefore, future research should not be limited to the DPMM ScheLoc Problem but should also include other ScheLoc Problems.

References

[1] D. Elvikis, H. W. Hamacher, and M. T. Kalsch. Simultaneous scheduling and location (scheloc): The planar scheloc makespan problem. *Journal of Scheduling*, 12:361–374, 2008.

[2] R. Graham. Bounds for certain multiprocessing anomalies. *Bell System Technical Journal*, 45:1563–1581, 1966.

[3] H. W. Hamacher and H. Hennes. Integrated scheuling and location models: Single machine makespan problems. *Studies in Locational Analysis*, 16:77–90, 2007.

[4] H. Hennes. *Integration of Scheduling and Location Models.* PhD thesis, University of Kaiserslautern, 2005.

[5] M. Kalsch and Z. Drezner. Solving scheduling and location problems in the plane simultaneously. *Computurs and Operations Research*, 37:256–264, 2010.

[6] M. T. Kalsch. *Scheduling - Location (ScheLoc) Models, Theory and Algorithms.* PhD thesis, University of Kaiserslautern, 2009.

[7] C. Kaufmann. A polynomial time algorithm for an integrated scheduling and location problem. In *Proceedings of the 14th International Conference on Project Management and Scheduling*, pages 124–128, 2014.

[8] F. E. Maranzana. On the location of supply points to minimize tranport costs. *Operational Research Quarterly*, 15:261–270, 1964.

[9] N. Mladenović, J. Brimberg, P. Hansen, and J. A. Moreno Pérez. The p-median problem: A survey of metaheuristic approaches. *European Journal of Operational Research*, 179:927–937, 2007.

[10] S. Nickel and J. Puerto. *Location Theory: A Unified Approach.* Springer Berlin, 2005.

[11] J. Reese. Solution methods for the p-median problem: An annotated bibliography. *NETWORKS*, 48:125–142, 2006.

[12] B. Tansel, R. Francis, and T. Lowe. State of the art - location on networks: A survey. part i: The p-center and p-median problems. *Management Science*, 29:482–497, 1983.

[13] S. Wesolkowski, N. Francetić, and S. C. Grant. Trade: Training device selection via multi-objective optimization. In *IEEE Congress on Evolutionary Computation*, 2014.

[14] R. Xu and D. Wunsch. Survey of clustering algorithms. *IEEE Transactions on Neural Networks*, 16:645–678, 2005.