The Partitioning Min-Max Weighted Matching Problem†

Dominik Kress, Sebastian Meiswinkel*, Erwin Pesch

Department of Management Information Science, University of Siegen, Kohlbettstr. 15, D-57068 Siegen, Germany

Abstract

We introduce and analyze the Partitioning Min-Max Weighted Matching (PMMWM) Problem. PMMWM combines the problem of partitioning a set of vertices of a bipartite graph into disjoint subsets of restricted size and the strongly NP-hard Min-Max Weighted Matching (MMWM) Problem, that has recently been introduced in the literature. In contrast to PMMWM, the latter problem assumes the partitioning to be given. Applications arise in the field of intermodal container terminals and sea ports. We propose a MILP formulation for PMMWM and prove that the problem is NP-hard in the strong sense. Two heuristic frameworks are presented. Both of them outperform standard optimization software. Our extensive computational study proves that the algorithms provide high quality solutions within reasonable time.

Keywords: Assignment, Partitioning, Maximum Matching, Bipartite Graph, Container Transshipment

1. Introduction

In this paper we consider a variant of the strongly NP-hard Min-Max Weighted Matching (MMWM) Problem, that has recently been introduced by Barketau et al. (2015). An instance of MMWM is defined by an edge-weighted bipartite graph $G(U, V, E)$ with disjoint vertex sets U and V (bipartitions), edge set E, and a partitioning of U into disjoint subsets (components). Given a maximum matching on G, the weight of a component is defined as the sum of the weights of the edges of the matching that are incident to the vertices of the component. The objective is to find a maximum matching that minimizes the maximum weight of the components. The components may, for

*Corresponding author, phone: +49 271 740 4597, fax: +49 271 740 2940

Email addresses: dominik.kress@uni-siegen.de (Dominik Kress), sebastian.meiswinkel@uni-siegen.de (Sebastian Meiswinkel), erwin.pesch@uni-siegen.de (Erwin Pesch)

† This is an Accepted Manuscript of an article published by Elsevier in the **European Journal of Operational Research**, available online: https://doi.org/10.1016/j.ejor.2015.06.041

example, correspond to areas of responsibility of managers or tasks to be performed by a worker or machine. The objective is to balance the workload, risk, etc. over these components.

While Barketau et al. (2015) assume the components to be fixed, we relax this assumption by assuming the partitioning decision to be part of the optimization, with only the desired number of components being fixed. We refer to this problem as the Partitioning Min-Max Weighted Matching (PMMWM) Problem. Figure 1 illustrates an exemplary solution to an example instance of PMMWM. The maximum matching is represented by bold edges. Edge weights are solely depicted for the edges of the matching. Bipartition $U = \{u_1, ..., u_7\}$ has been partitioned into the components U_1, U_2 and U_3 with weights 6, 9 and 4. Hence, the corresponding objective function value of the PMMWM instance is $\max\{6, 9, 4\} = 9$. If we move u_4 to U_3 without changing the matching, the objective function value reduces by 1.

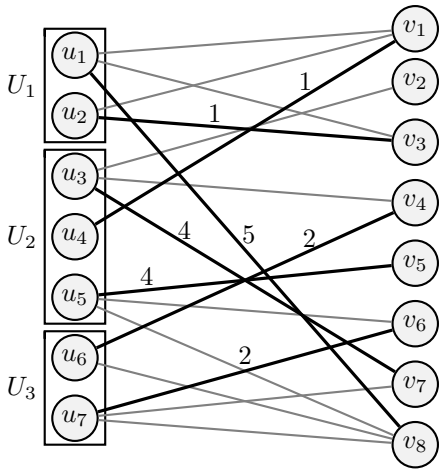

Figure 1: A solution to an example instance of PMMWM.

This paper is organized as follows. In Section 2 we provide a detailed problem description along with a MILP model of the problem. We present two applications of PMMWM in the context of a reach stacker based container terminal and a rail-road terminal. Next, a proof of the problem's strong NP-hardness is given in Section 3. Section 4 introduces two heuristic frameworks that are being analyzed based on computational tests in Section 5. In Section 6, we summarize the findings of this paper.

2. Detailed Problem Definition and Applications

Let $G(U, V, E)$ be a weighted bipartite graph with bipartitions U and V and edge set E. The elements of U and V are indexed $i = 1, \ldots, n_1$ and $j = 1, \ldots, n_2$, respectively.

Assume $n_1 \leq n_2$. A weight $c(e) = c_{uv} \in \mathbb{Q}_0^+$ is associated with each edge $e = (u, v) \in E$ of G. Define a matching as a set $M \subseteq E$ of pairwise nonadjacent edges and a maximum matching as a matching having the largest possible size $|M|$ amongst all matchings on G. Throughout the paper, we will assume that, for any given bipartite graph, there exists a maximum matching Π with $|\Pi| = n_1$. As in Barketau et al. (2015), given a partitioning of U into m disjoint subsets, $U_1, U_2, \ldots, U_m$, the value of a maximum matching Π is defined to be $w(\Pi) := \max_{k\in\{1,\ldots,m\}}\{\sum_{u\in U_k,(u,v)\in\Pi} c_{uv}\}$ (refer to Figure 1 for an illustration). Then PMMWM can formally be defined as follows: Find a partitioning of the vertex set U into m (potentially empty) disjoint subsets, $U_1, U_2, \ldots, U_m$, with at most $\bar{u}$ elements in each subset, and a maximum matching Π on G, such that the value of Π is minimum amongst all maximum matchings over all possible partitionings of U.

We define the following binary variables:

$$x_{ij} := \begin{cases} 1 & \text{if } (i,j) \in \Pi, \\ 0 & \text{else,} \end{cases} \qquad \forall\, (i,j) \in E \tag{1}$$

and

$$y_{ik} := \begin{cases} 1 & \text{if } i \in U_k, \\ 0 & \text{else,} \end{cases} \qquad \forall\, i \in U, k \in \{1,\ldots,m\}. \tag{2}$$

Then a nonlinear mathematical model for PMMWM is as follows:

$$\min_{\mathbf{x},\mathbf{y}} \quad \max_{k\in\{1,\ldots,m\}} \left\{ \sum_{i\in U} \sum_{j\in V} c_{ij} y_{ik} x_{ij} \right\} \tag{3}$$

$$\text{s.t.} \quad \sum_{j\in V} x_{ij} = 1 \qquad \forall\, i \in U, \tag{4}$$

$$\sum_{i\in U} x_{ij} \leq 1 \qquad \forall\, j \in V, \tag{5}$$

$$\sum_{k=1}^{m} y_{ik} = 1 \qquad \forall\, i \in U, \tag{6}$$

$$\sum_{i\in U} y_{ik} \leq \bar{u} \qquad \forall\, k \in \{1,\ldots,m\}, \tag{7}$$

$$x_{ij} \in \{0,1\} \qquad \forall\, (i,j) \in E, \tag{8}$$

$$y_{ik} \in \{0,1\} \qquad \forall\, i \in U,\, k \in \{1,\ldots,m\}. \tag{9}$$

The objective function (3) minimizes the value of the maximum matching over all possible

partitionings and all maximum matchings. Constraints (4)–(5) are well known maximum matching constraints (recall that $n_1 \leq n_2$). Constraints (6) enforce every vertex $u \in U$ to be an element of exactly one partition U_k, $k \in 1, \ldots, m$. Constraints (7) restrict the number of vertices in each partition to be at most $\bar{u}$. The domains of the variables are defined by (8)–(9).

A specific application of PMMWM arises at small to medium sized sea ports where containers are handled by reach stackers. The corresponding terminals, as schematically represented in Figure 2, can include large long-term storage areas and additional tem-

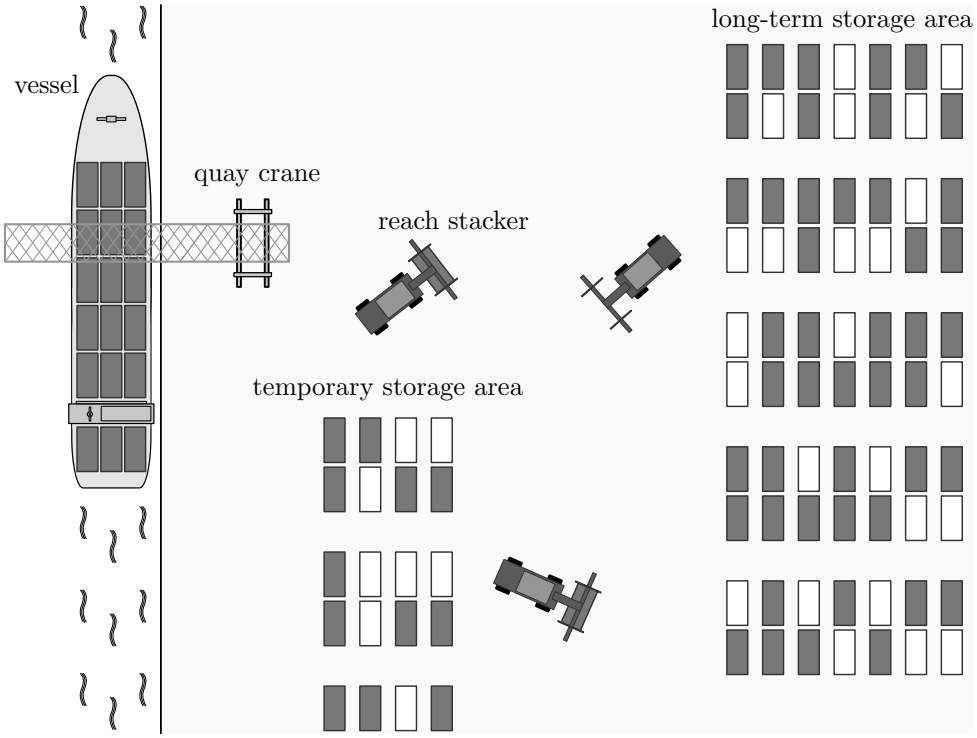

Figure 2: Potential schematic layout of a reach stacker based terminal.

porary storage areas (or marshaling-areas; see, for instance, Kozan & Preston, 1999; Preston & Kozan, 2001). The latter areas aim at improving the performance of the terminals by inducing short turnaround times of vessels when distances to long-term storage areas are relatively large. When a vessel arrives at a berth at the terminal, containers are unloaded by quay cranes and then stored in a temporary storage area that is located next to the berth. Containers that leave the terminal by ship are moved to the temporary storage area using reach stackers during previous idle times. We will assume that these vehicles are "fast" if they are unloaded and "slow" if they are loaded and can thus restrict ourselves to considering the movements of loaded vehicles only. This is a common assumption when considering container movements (see, for instance, Boysen

& Fliedner, 2010) and is supported by the fact, that "reach stackers [in comparison to straddle carriers] are less stable in the forward direction as the machines will fall forward when breaking in an emergency, particularly if the load is carried high for visibility reasons" (Isoloader, 2012). Then an application of PMMWM arises, when considering the process of emptying or refilling the temporary storage area during idle times. We are faced with the problem of assigning each container in the temporary storage area to an empty slot in the long-term storage area (or vice versa) and assigning the corresponding container movements to a limited number of available vehicles. Consider, for example, the case of filling the temporary storage area with containers of the long-term storage area. The vertex set U corresponds to the set of containers that have to be moved. Bipartition V relates to the set of empty slots in the temporary storage area. The edge weights c_{uv} include multiple effects. First, they obviously result from the distances between container locations and their potential temporary storage slots. Additionally, the weights include a (potentially container dependent) amount of time needed to lift and drop a container by a reach stacker. Furthermore, it is possible to represent different velocities of loaded vehicles, for example depending on the movement of heavy, sensitive, or empty containers. The parameter m represents the number of available reach stackers. $\bar{u}$ represents an upper bound on the maximum number of containers that a reach-stacker may process. A solution is on one hand composed of an assignment of each container to an empty storage slot, i.e. a maximum matching, and on the other hand of an assignment of container movements to the available vehicles, i.e. a partitioning of U into at most m components. Naturally, the objective is to perform all movements as fast as possible by balancing the workload over the available reach stackers.

Applications of PMMWM may also arise when considering container transshipment in rail-road terminals (cf. Boysen et al., 2013, for a survey) and can thus be motivated in analogy to Barketau et al. (2015), who point out that an "interesting direction of [...] research is the case when the components are not fixed and the decision on their sizes is a part of the problem." A schematic representation of a classical layout of such terminals is given in Figure 3 (Boysen & Fliedner, 2010; Boysen et al., 2013). The terminal consists of a given number of parallel railway tracks, a container storage area (intermediate storage) including a fully automated sorting system, and truck lanes. Each of these areas is subdivided into segments, usually referred to as slots, that are adjusted to the length of a standardized railcar. Multiple gantry cranes, spanning over these areas,

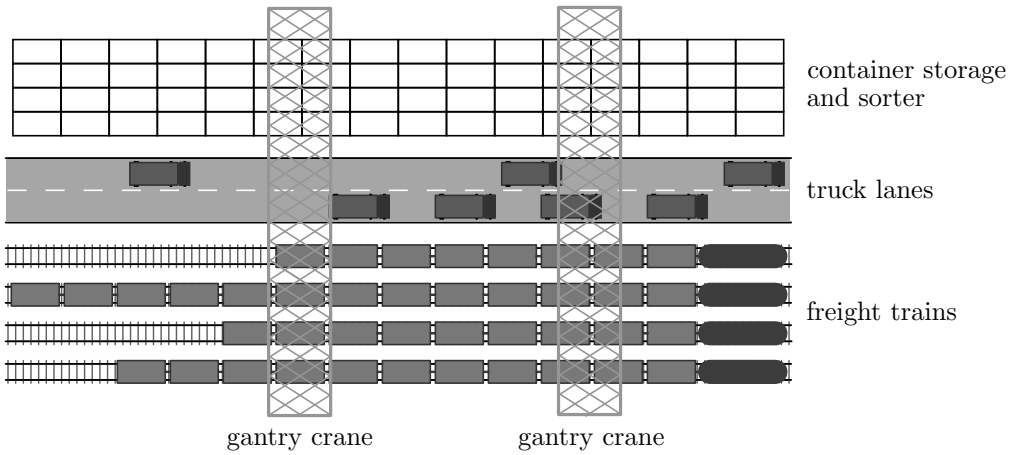

Figure 3: Rail-road terminal (Boysen & Fliedner, 2010).

transfer containers between trucks and railcars. In mathematical models, workloads of the cranes are typically approximated by only considering laden movements. Trains are usually served in bundles. A bundle leaves the terminal after all corresponding transshipment operations have been performed. In a classical setting, the gantry cranes may not cross each other. However, one can easily think of a potential (not yet existent) layout of rail-road terminals with gantry cranes being able to pass one another, as such systems can, for example, frequently be found at sea terminals (Kemme, 2012). To extract the most basic problem setting, we assume that these cranes are always able to pass one another (as opposed to modelling more complicated passing rules). In such a case, an application of PMMWM is straight forward: Given a set U of containers to be transferred to potential target slots V by m gantry cranes, one has to find a specific assignment of containers to slots, i.e. a maximum matching, and a partitioning of U into m components representing the container movements to be performed by the different cranes, such that the workload of the cranes is "balanced". Obviously, the edge weights of the corresponding bipartite graph represent workloads of laden movements. Let us now consider the classical layout of rail-road terminals, i.e. the case of the gantry cranes not being able to pass one another. One way of handling the non-passing restrictions is to horizontally subdivide the terminal into crane areas, each being exclusively processed by one of the gantry cranes. Any container transport between the areas is performed using the sorter. The planning process at such rail-road terminals can then be thought of as a sequential process. Two subsequent decisions in this process are concerned with the formation of crane areas followed by the assignment of containers to target slots, where a container stored in a specific crane area has to be operated by the area-specific

crane to perform the movement of the container to its assigned target slot. When adding additional restrictions, PMMWM integrates these decisions, with containers U having to be partitioned into m (number of gantry cranes) components (i.e. crane areas), and target slots V having to be assigned to the containers (maximum matching). Again, the edge weights of the corresponding bipartite graph represent the cost of laden movements and we aim for the workloads of the cranes being "as similar as possible" (minimax objective of PMMWM). However, additional restrictions will, for example, need to enforce elements of partitions to be "neighbored" (i.e. crane areas to be connected). Additionally, appropriate penalty costs will have to account for transferring containers between crane areas using the sorter.

We close this section by noting that model (3)–(9) can easily be linearized by introducing additional variables $z_{ijk} \geq 0$ for all $i \in U$, $j \in V$ and $k \in \{1, \ldots, m\}$:

$$\min_{\mathbf{x},\mathbf{y}} \quad c \tag{10}$$

$$\text{s.t.} \quad \text{constraints (4)–(9)},$$

$$c \geq \sum_{i \in U} \sum_{j \in V} c_{ij} z_{ijk} \qquad \forall\, k \in \{1, \ldots, m\}, \tag{11}$$

$$z_{ijk} \geq y_{ik} + x_{ij} - 1 \qquad \forall\, i \in U,\, j \in V,\, k \in \{1, \ldots, m\}, \tag{12}$$

$$z_{ijk} \geq 0 \qquad \forall\, i \in U,\, j \in V,\, k \in \{1, \ldots, m\}. \tag{13}$$

3. Computational Complexity

It is easy to see that if we fix m to one in PMMWM, i.e. if we consider the case of having exactly one component, we are left with the well-known problem of finding a maximum matching of minimum weight in a bipartite graph, which is equivalent to the Linear Assignment Problem and can thus be solved in polynomial time (see Section 4). The special case of PMMWM where the number of components m is equal to $|U|$ is known as the Linear Bottleneck Assignment Problem (cf. Burkard et al., 2009, and the references therein). It is polynomially solvable as well. In particular, it can be solved by the threshold method in $O(n^2\sqrt{n/\log n})$ time. The Weighted Edge Dominating Set Problem is similar to PMMWM with one component but for arbitrary graphs. It is NP-hard and admits a 2-approximation.

Now note that PMMWM is not a straight forward generalization of MMWM in the

sense that, given an instance I of MMWM, we need only "copy" the corresponding bipartite graph, fix a set of PMMWM parameters to specific values, and solve the resulting PMMWM instance to receive an optimal solution to I. More generally, it is not trivial to construct an instance of PMMWM that (when solved to optimality) is guaranteed to result in the partitioning given in I and thus to provide an optimal solution to I. Thus, MMWM is not simply a subproblem of PMMWM. In MMWM the components of U are fixed and are not necessarily equal in size while PMMWM only imposes an upper bound on the size of the components. Moreover, the fact that the number and the elements of the (non empty) components of U are a result of the optimization consequently leads to a higher flexibility and a larger search space that does not necessarily contain all solutions of MMWM (even if m is chosen appropriately), unless $\bar{u}$ is sufficiently large. Thus we cannot simply conclude that PMMWM is NP-hard from the NP-hardness of MMWM.

A formal proof of strong NP-hardness of PMMWM, which we will now provide, is more naturally based on a reduction of a classical partitioning problem than on a matching problem.

We will consider the decision problem related to PMMWM:

Definition 3.1 (PMMWM-D)**.** Given a weighted bipartite graph as defined in Section 2. Does there exist a partitioning of the vertex set U into m (potentially empty) disjoint subsets, $U_1, U_2, \ldots, U_m$, with at most $\bar{u}$ elements in each subset, and a maximum matching Π on G, such that the value of Π is no larger than a given $\omega \in \mathbb{Q}^+$?

The proof is based on a reduction of the strongly NP-complete 3-Partition Problem (see Garey & Johnson, 1979):

Definition 3.2 (3-Partition)**.** Given a set A of $3k$ elements, a bound $B \in \mathbb{Z}^+$, and sizes $s(a) \in \mathbb{Z}^+$ for all $a \in A$ such that $B/4 < s(a) < B/2$ and such that $\sum_{a \in A} s(a) = kB$. Is there a partitioning of A into k disjoint subsets $A_1, \ldots, A_k$ such that, for $1 \leq i \leq k$, $\sum_{a \in A_i} s(a) = B$?

Theorem 3.1. *PMMWM-D is NP-complete in the strong sense.*

Proof. Obviously PMMWM-D $\in$ NP because for any partitioning and maximum matching it can be checked in polynomial time if the corresponding value is no larger than ω.

Now consider an arbitrary instance I of 3-PARTITION and construct a complete bipartite graph G as an instance J of PMMWM-D with $U = A$ and vertex set V such that $|V| = |U|$. Furthermore, for each $u \in U$, define the weights of all edges incident to u in G to be equal to the corresponding element's size $s(a)$, $a \in A$. Set $m = k$, $\bar{u} = 3$, and $\omega = B$. Now note that, given an arbitrary partitioning $U_1, \ldots, U_m$ of U, every maximum matching has the same value. We are left with the task of showing that there exists a partitioning $U_1, \ldots, U_m$ of U and a maximum matching Π with a value of no more than ω if and only if the answer to I is yes (we will say that I "has a solution" if its answer is yes and call a corresponding partitioning of A a "solution").

Suppose we are given a solution $A_1, \ldots, A_k$ to I. Construct a partitioning $U_1, \ldots, U_m$ of U such that subset U_i contains all vertices that correspond to elements of A_i for all $i = 1, \ldots, m$, and consider any maximum matching on G. It is immediately implied that the value is ω.

Suppose we are given a partitioning $U_1, \ldots, U_m$ of U and a maximum matching Π such that its value is no more than ω. As $\bar{u} = 3$ and $|U| = 3m$, we necessarily have $|U_i| = 3$ for all $i = 1, \ldots, m$. Now note that, as the sum of the edge weights of any maximum matching on G equals $m\omega$ and the value of Π is at most ω, we have $\sum_{u \in U_i, (u,v) \in \Pi} c_{uv} = \omega$ for all $i = 1, \ldots, m$. Hence, the subsets of A that correspond to the given partitioning of U establish a solution to I. □

We conclude:

Corollary 3.1. *PMMWM is NP-hard in the strong sense.*

4. Algorithms

PMMWM can be considered as a combination of a matching and a partitioning problem. Hence, a natural way of constructing heuristics for PMMWM is to decompose the problem into its matching and partitioning components, solve the resulting problems separately (either exact or heuristically), and combine the resulting solutions to a solution of PMMWM. Obviously, we can construct different heuristics by changing the order of solving the separate stages.

Partition-Match heuristics assume that there is a vertex weight associated with each element of vertex set U and, in the first stage, partition U into no more than m components with at most $\bar{u}$ elements in each component, such that the maximum weight of

the components is as small as possible. Here, the weight of a component is defined as the sum of the weights of the vertices of the component. When considering the objective of minimizing the maximum weight of the components, we refer to this problem as the Restriced Partitioning (RP) Problem.

We note:

Theorem 4.1. *RP is NP-hard in the strong sense.*

As the proof of Theorem 4.1 is in analogy to the proof of Theorem 3.1 and Corollary 3.1, we will not present it in detail for the sake of brevity.

In the next stage of a Partition-Match heuristic, we drop the weights of the vertices. Given the components resulting from the first stage, we now need to determine a maximum matching of small value. Hence, we are faced with an instance of MMWM. Solving this problem results in a maximum matching, that we may use for restarting the overall procedure by solving RP with each vertex weight being equal to the weight of the edge of the maximum matching that is incident to the very vertex. Figure 4 illustrates the idea of Partition-Match heuristics.

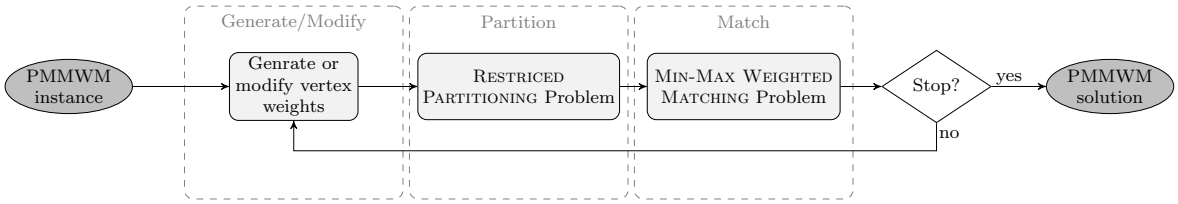

Figure 4: Partition-Match heuristics.

Match-Partition heuristics (see Figure 5) first consider the classical problem of finding a maximum matching of minimum weight, i.e. minimum sum of the weights of all edges of the matching, which we refer to as the Min-Sum Weighted Matching (MSWM) Problem. MSWM can be solved in polynomial time, for example in $\mathcal{O}(n^3)$ (where $n := \max\{n_1, n_2\}$) by applying the well known Hungarian algorithm (cf. Burkard et al., 2009; Kuhn, 1955). Hereafter, we proceed in analogy to our approach in Partition-Match heuristics, i.e. by defining vertex weights based on the maximum matching and solving RP. Finally, we modify the edge weights of the bipartite graph to restart the procedure by solving MSWM on the modified graph.

We will present details on the algorithms in the next subsections. Subsection 4.1 is concerned with solving RP. Heuristics for solving MMWM are presented in Subsection

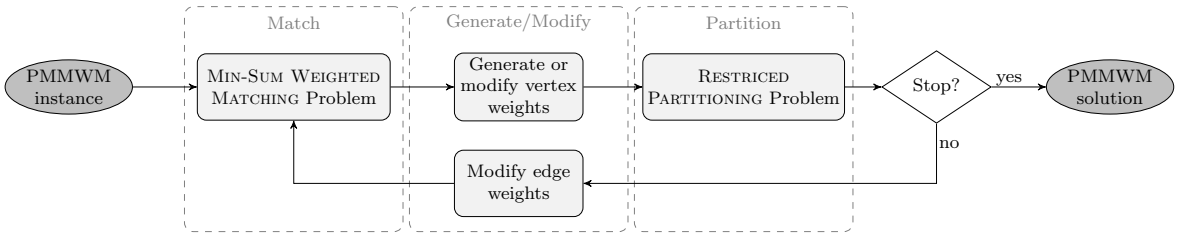

Figure 5: Match-Partition heuristics.

4.2. Hereafter, we will present details of Partition-Match and Match-Partition heuristics in Subsections 4.3 and 4.4.

4.1. Solving the Restricted Partitioning Problem

As stated in Theorem 4.1, RP is NP-hard in the strong sense. We therefore apply a heuristic algorithm when facing an instance of RP in Partition-Match or Match-Partition heuristics. As RP is similar to the well known Multiprocessor Scheduling Problem, that has been extensively studied in the literature (see, for example, the literature review by Chen et al., 1999), our approach adapts ideas of Lee & Massey (1988), who present a heuristic for the latter problem, in its first stage.

An instance of RP is defined by a set U, $|U| = n_1$, of elements (vertices) with weights $w_i \in \mathbb{Q}^+$, $i \in U$, an upper bound $\bar{u}$, corresponding to the maximum number of elements in each component, and an upper bound m, defining the maximum number of components. The objective is to find a feasible partitioning of U, such that the maximum weight (as defined above) of the components is minimized. A partitioning is feasible, if and only if it consists of no more than m components with each component containing at most $\bar{u}$ elements.

Our heuristic, denoted by RPH, is composed of two stages. First, we construct a feasible solution as follows: Let U' be the elements of U sorted in non-increasing order of their weights and initialize m empty components $U'_1, \ldots, U'_m$. Select an element with largest weight from U' and add it to a component with the smallest weight among all components with less than $\bar{u}$ elements. The stage ends when U' is empty.

The next stage is a local search on the obtained solution $(U'_1, \ldots, U'_m)$. Denote the weight of component U'_i, $i \in \{1, \ldots, m\}$, by $w(U'_i)$. Select $U'_{min} \in \arg\min_{U'_i,\, i\in\{1,\ldots,m\}} w(U'_i)$ and $U'_{max} \in \arg\max_{U'_i,\, i\in\{1,\ldots,m\}} w(U'_i)$ and sort the elements of these sets in non-decreasing order and non-increasing order of their weights, respectively. Let u_i^{max} be the i-th element of the sorted component U'_{max} and u_i^{min} the i-th element of the sorted component

U'_{min}. Next, select one element of U'_{max} after another, starting with u_1^{max}, and interchange it with a set $S = \{u_i^{min}, \ldots, u_j^{min} | 1 \leq i \leq j \leq |U'_{min}|\}$ of elements of component U'_{min}, if the interchange results in feasible components and reduces the difference $|w(U'_{max}) - w(U'_{min})|$. If an interchange has been performed, restart the local search. Otherwise, if we cannot find a suitable set S for any element of U'_{max}, stop the local search.

4.2. Solving the Min-Max Weighted Matching Problem

We apply two heuristics for solving instances of MMWM. First, we implemented an approximate algorithm that has been developed in Barketau et al. (2015) when introducing MMWM. We will refer to this algorithm as the BPS heuristic. Basically, the algorithm enumerates over a set of m multipliers that are applied to modify edge weights of the bipartite graph. For each modified problem instance, the corresponding MSWM instance is solved by applying the Hungarian algorithm. The resulting matching is then used to compute the corresponding objective function value of MMWM. BPS then chooses the best three solutions and tries to improve them by a local search procedure. We will refer to this local search procedure as LS. For details of the implementation, we refer to Barketau et al. (2015).

While Barketau et al. (2015) show BPS to be an adequate method for solving MMWM alone, we implemented another, rather simple, heuristic. This is motivated by the fact that, when calling a Partition-Match heuristic, we have to solve multiple instances of MMWM. Hence, in order to reduce running times of the overall procedure, a simple heuristic seems to be promising. As the approach is based on the regret principle (cf., for instance, Domschke & Scholl, 2005), we refer to it as the REG heuristic. We say that an edge belongs to a component, if it is incident to a vertex of the (given) component.

REG starts by initializing an empty matching $M = \emptyset$. For each component $i = 1, \ldots, m$ of the partitioning, REG then determines the cheapest and second-cheapest edge, e_1^i and e_2^i, amongst the edges of the component that are not yet incident to matched vertices of the bipartite graph. The difference of the related edge weights determines the regret value $reg_i = c(e_2^i) - c(e_1^i)$ for each component $i = 1, \ldots, m$. If all but one (e_1^i) edges of a component $i \in \{1, \ldots, m\}$ are incident to a matched vertex, set $reg_i = L - c(e_1^i)$, where L is a sufficiently large number. If no edge of a component $i \in \{1, \ldots, m\}$ remains

incident to an unmatched vertex, set $reg_i = -1$. Next, choose a component c with the largest regret amongst all components and add edge e_1^c to M. Repeat the procedure until $reg_i = -1$ for $i = 1, \ldots, m$. If the corresponding matching is not maximum, determine an augmenting path and augment M by applying the procedure described in Burkard et al. (2009, Chapter 4, Algorithm 4.2). Hereafter, if M is maximum, then stop. Otherwise continue with computing regret values.

4.3. Partition-Match Heuristics

To initialize a Partition-Match heuristic (see Figure 4), we generate (vertex) weights that are necessary for partitioning the vertex set U by RPH (Section 4.1). We tested several strategies of generating these weights. As they all performed similarly in computational tests, we decided on applying the average weight of all edges incident to vertex u for each vertex $u \in U$.

Once we have generated the initial weights for each vertex $u \in U$, we enter the main loop of Figure 4. As described above, this loop consists of three stages. First, solve RP based on the vertex weights that have been generated. Here, we apply RPH. Second, solve MMWM based on the resulting partitioning. We tested both of the heuristics described in Section 4.2, i.e. BPS and REG, to solve MMWM. When calling REG, we apply the local search procedure LS (see Section 4.2) to potentially improve the solution found by REG (note that BPS applies LS to three solutions). Third, generate new vertex weights based on the current maximum matching as described above. To avoid cycling of the algorithm, we monitor the best known solution of the PMMWM input-instance, being composed of the solutions of RP and MMWM. If there has been no improvement for 20 iterations of the main loop, we stop the Partition-Match algorithm.

We will refer to the Partition-Match heuristic as described in this section as PM_{BPS} when applying BPS. When calling REG (including LS), we will refer to the resulting Partition-Match heuristic as PM_{REG}.

4.4. Match-Partition Heuristics

As described above, Match-Partition heuristics differ from Partition-Match heuristics in the subproblems that need to be solved. Again, the main loop (see Figure 5) is composed of three stages. First, we transform the PMMWM input-instance into an instance of MSWM by dropping the partitioning constraints. To solve the resulting

instance of MSWM, we apply a $\mathcal{O}(n^3)$ (again $n := \max\{n_1, n_2\}$) version of the Hungarian algorithm as presented in Burkard et al. (2009). Given the matching generated in the first stage, we generate an instance of RP by assigning to each vertex $u \in U$ the weight of the edge incident to u in the obtained matching. As before, the combination of the solutions of the first two stages define a solution of the PMMWM input-instance. As we may apply LS (see Section 4.2) to potentially improve this solution, we will refer to our Match-Partition heuristic as MP if LS is not used and MP_{LS} if LS is used. If the resulting solution to PMMWM has not improved for 20 iterations, we stop the Match-Partition heuristic. In the third stage, we alter the weights of the edges of the bipartite graph. We apply a simple transformation strategy, that increases the weight of exactly one edge $e \in E$ in each iteration of the main loop. The selection of edge e depends on the current solution of the PMMWM input-instance: Select the component with the largest weight. Among the current edges of the matching that are incident to vertices of this component, select the one with largest weight. Then, set the weight of this edge to a large value (we set the weight to $10^2 \cdot c_{\max}$, where $c_{\max}$ is the largest edge weight of the input graph, in our computational tests) to reduce the probability that it will be part of the solution of the altered MSWM instance. After λ iterations of the main loop, restore the original edge weight of edge e. Our computational tests show that $\lambda = \lceil 0.1 \cdot n_1 \cdot n_2 \rceil$ is a reasonable value. Note that the perturbed edge weights are only used for the matching problem in stage one. The vertex weights assigned in stage two are based on the original weights of the input graph.

5. Computational Results

In order to assess the performance of the algorithms introduced in Section 4, we ran extensive computational tests. All computational tests were performed on an Intel Core i7 mobile CPU at 2.8GHz and 8GB of RAM, running Windows 7 64bit. All algorithms were implemented in C^{++} (Microsoft Visual Studio 2010). We used IBM ILOG CPLEX in version 12.5 with 64bit.

In order to get a comprehensive overview of the performance of our algorithms, we use different strategies of generating instances. All instances are defined on bipartite graphs with $n_1 = n_2$, since the case $n_1 \neq n_2$ trivially reduces to the case $n_1 = n_2$ by adding edges of zero weight. We define $n := n_1 = n_2$.

The first strategy of generating instances is inspired by Barketau et al. (2015). We

refer to the instance sets generated by this strategy as BPS70 and BPS80. We generate n^2 rational numbers uniformly distributed in the interval $[1, 1000]$ and a complete bipartite graph with vertex sets $U = \{u_1, \ldots, u_n\}$ and $V = \{v_1, \ldots, v_n\}$. We sort the numbers in non-decreasing order and denote the resulting list by L. Hereafter, we consecutively assign these numbers (as edge weights) to the edges in the following manner: We start with vertex v_1 and assign the first $\lfloor 0.8n \rfloor$ (in case of BPS80) or $\lfloor 0.7n \rfloor$ (in case of BPS70) elements of L to the edges $(u_1, v_1), (u_2, v_1), \ldots, (u_{\lfloor 0.8n \rfloor}, v_1)$ (BPS80) or $(u_1, v_1), (u_2, v_1), \ldots, (u_{\lfloor 0.7n \rfloor}, v_1)$ (BPS70) and remove them from L. For the remaining edges being incident to vertex v_1, we randomly choose and assign elements of L. Once an element of L has been assigned, we remove it from L. The procedure is repeated for all remaining vertices $v \in V$, $v \neq v_1$.

The second strategy of generating instances constructs complete bipartite graphs and randomly determines and assigns edge weights uniformly distributed in the interval $[1, 1000]$. We refer to instances generated by this strategy as RAND.

Besides BPS70, BPS80, and RAND, all of which are based on complete bipartite graphs, we generate instances defined on non-complete (or sparse) bipartite graphs with $|E| = \lceil 0.7n^2 \rceil$ or $|E| = \lceil 0.8n^2 \rceil$. We refer to them as SPARSE70 and SPARSE80, respectively. When generating the edges, we guarantee that there exists at least one feasible solution for the resulting PMMWM instance (cf. restrictions (4)). Again, edge weights are determined randomly and uniformly distributed in the interval $[1, 1000]$.

For each strategy of generating instances, BPS70, BPS80, RAND, SPARSE70, and SPARSE80, we construct instances based on different parameter settings. We vary the size of the bipartitions from 10 to 190, i.e. $n = 10, 30, \ldots, 190$. Furthermore, we set m to 2, and (if larger than 2) to $\lfloor 0.04n \rfloor$, $\lfloor 0.08n \rfloor$ or $\lfloor 0.125n \rfloor$. Similarly, we set $\bar{u}$ to $\lceil \frac{n}{m} \rceil$, $\left\lfloor \lceil \frac{n}{m} \rceil + \frac{1}{3} \left(n - \lceil \frac{n}{m} \rceil \right) \right\rfloor$, or n. For each combination of parameters and each generation strategy, we construct five PMMWM instances. This results in a total of 2,475 instances.

Let F^* be the value of the objective function (10) of the best solution found by a specific heuristic. Then, we rate the quality of this heuristic with the ratio $\frac{F^*}{F^{best}}$, where F^{best} is the best objective function value among the objective function values of the solutions obtained by any of the heuristics PM_{REG}, PM_{BPS}, MP and MP_{LS} and the best solution found by CPLEX (model (10)–(13)) within a time limit of 180 seconds.

Figure 6 gives an overview of the algorithms' overall qualities over the different values

of n. For each n, the figure depicts the average quality over the results of all parameter settings and all generation strategies. Except for PM_{REG}, all heuristics perform very well, even for large values of n.

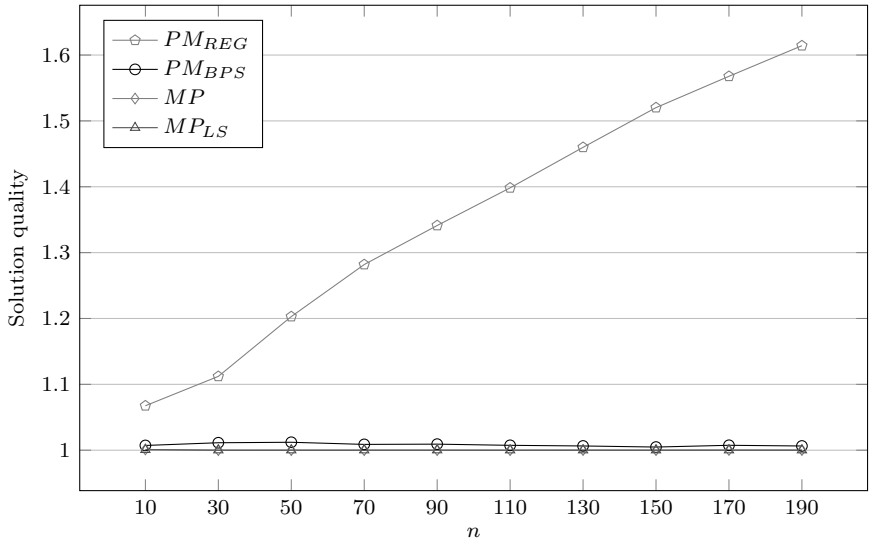

Figure 6: Overview of all heuristics.

As in Figure 6, Figure 7 presents a detailed view on the quality of PM_{BPS}, MP, and MP_{LS}. We find that both, MP and MP_{LS}, outperform PM_{BPS} in terms of solution quality. Note, however, that PM_{BPS} still performs at a high level. The specific effect of including (or not including) LS into Match-Partition heuristics, i.e. MP or MP_{LS}, will be analyzed in Figures 13 and 14.

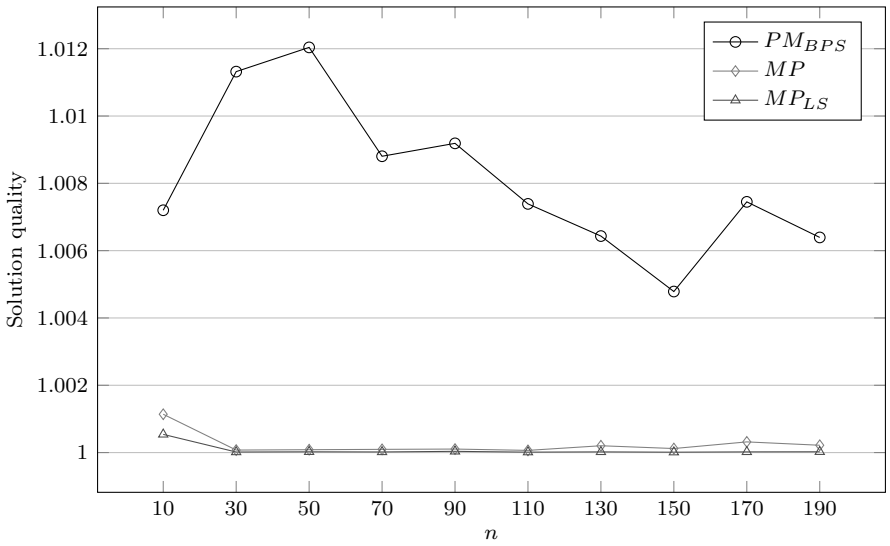

Figure 7: Comparison of MP_{LS}, MP, and PM_{BPS}.

Figures 8 and 9 compare the runtimes of the different heuristics. While, for all heuristics, runtimes increase roughly quadratically in n, we find that PM_{BPS} requires

by far the most computational effort. This fact stands against the gap in qualities of PM_{REG} and PM_{BPS}. PM_{REG}, MP, and MP_{LS} are quite similar in terms of runtimes.

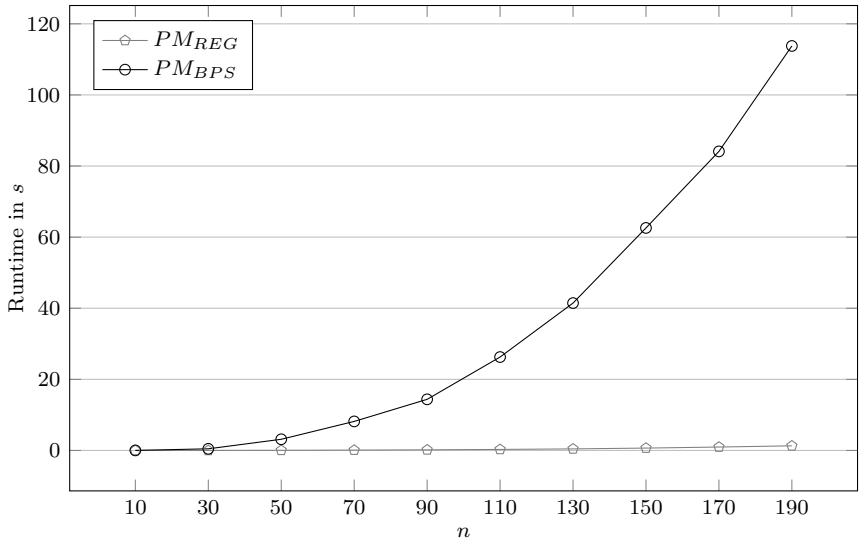

Figure 8: Runtimes of PM_{BPS} and PM_{REG}.

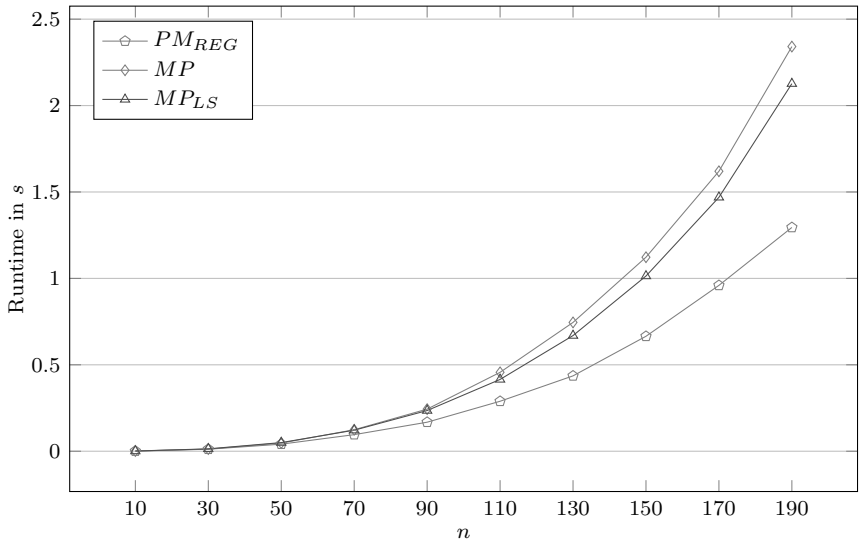

Figure 9: Runtimes of PM_{REG}, MP, and MP_{LS}.

In what follows, we will focus on the performance of CPLEX and the heuristics with respect to the different generation strategies, i.e. different structures of the underlying bipartite graph.

Figure 10 depicts the performance of CPLEX. Preliminary tests proved CPLEX to perform better for the linearized model (10)–(13) than for directly feeding it with model (3)–(9). Hence, we restrict ourselves to the linearized model. As mentioned above, CPLEX is stopped after 180 seconds and returns the best solution found within this time limit. Only instances with a size of $n = 10$ could be solved to optimality within this time limit. In general, Figure 10 shows the objective function values to be not competitive

when compared with the ones determined by the heuristics, which is especially obvious for $RAND$, $SPARSE70$, and $SPARSE80$ instances. The results for $BPS70$ and $BPS80$ instances are better. However, the objective function values determined by CPLEX are (on average) still up to 600% larger than the ones determined by the best heuristic (in case of $n = 190$).

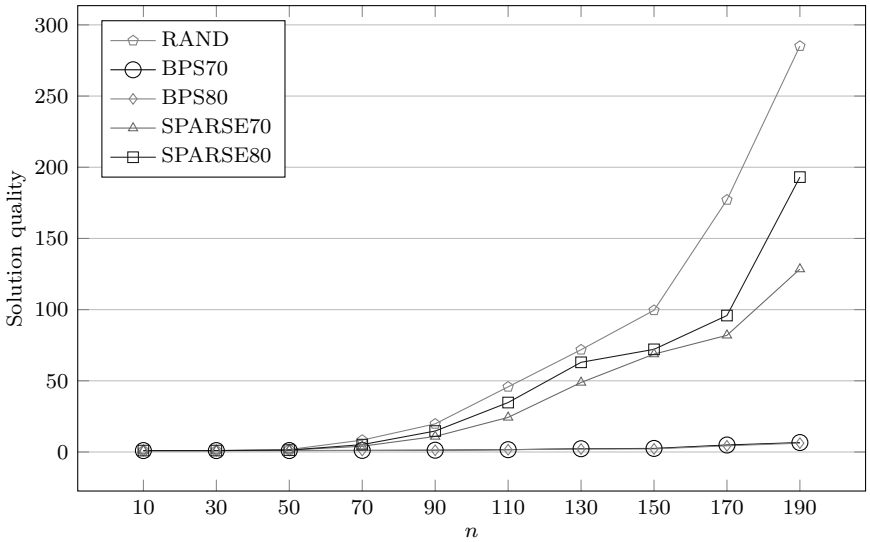

Figure 10: Quality of CPLEX.

Figure 11 provides results for PM_{REG}. The average quality of PM_{REG} is dramatically better than the average quality of CPLEX (Figure 10). While CPLEX reaches quality ratios larger than 250, these ratios are strictly smaller than 2.5 in case of PM_{REG}. Both, CPLEX and PM_{REG}, perform best on $BPS70$ and $BPS80$ instances. In case of PM_{REG}, the results for $BPS70$ and $BPS80$ are, on average, very close to the best solutions of all considered heuristics.

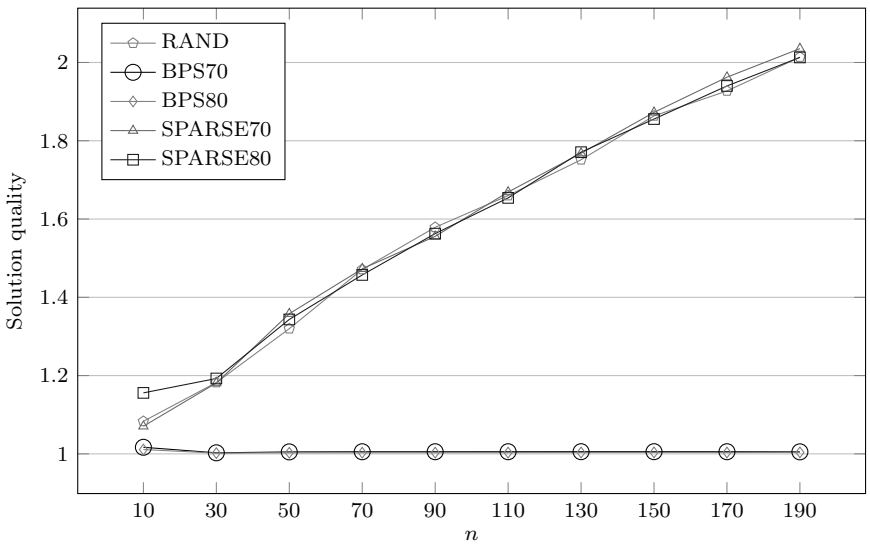

Figure 11: Quality of PM_{REG}.

The results shown in Figure 12 relate to PM_{BPS}. As before, PM_{BPS} is less effective in case of $RAND$, $SPARSE70$, and $SPARSE80$ instances, but very competitive in case of $BPS70$ and $BPS80$ instances. When comparing PM_{BPS} and PM_{REG}, keep in mind, that the runtimes of PM_{BPS} increase noticeably faster with increasing size of the bipartitions (Figure 8). For PM_{BPS} and PM_{REG}, increasing values of n seem to have a stronger effect on computational effort (in terms of runtime) than on solution quality.

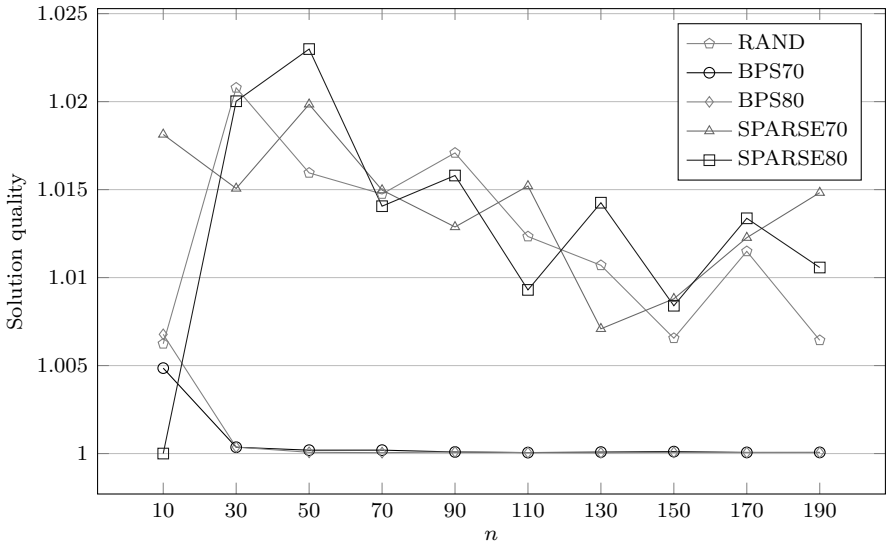

Figure 12: Quality of PM_{BPS}.

Finally, Figures 13 and 14 present results for MP and MP_{LS}. While MP seems to be less effective for $BPS70$ and $BPS80$ instances, MP_{LS} does not have this handicap. The peak of the graphs for $n = 10$ results from CPLEX being able to solve all related instances to optimality. Note that even in this case of comparison with optimal solutions, the quality ratio of MP_{LS} is less than 1.0011. For all considered values of n, the runtime of MP_{LS} is smaller than 6 seconds. The average runtime for $n = 190$ is around 2.2 seconds (see Figure 9). The influence of including LS on runtimes is (on average) less than a second for all n and all types of instances. Thus it is advisable to always apply MP_{LS} when facing a random instance of PMMWM.

6. Conclusion

We have introduced and analyzed the Partitioning Min-Max Weighted Matching (PMMWM) Problem. PMMWM is a combination of the problem of partitioning a set of vertices into disjoint subsets of restricted size and the strongly NP-hard Min-Max Weighted Matching (MMWM) Problem, that has recently been introduced by Barketau

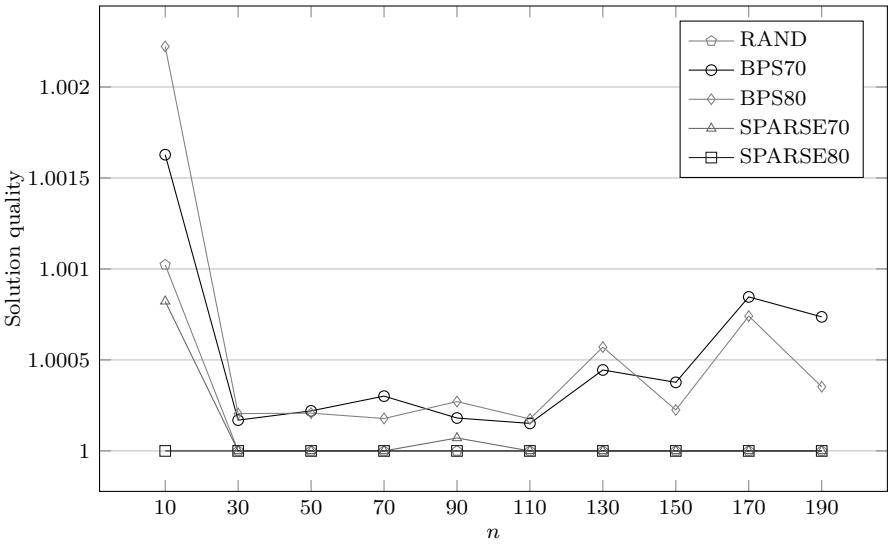

Figure 13: Quality of MP.

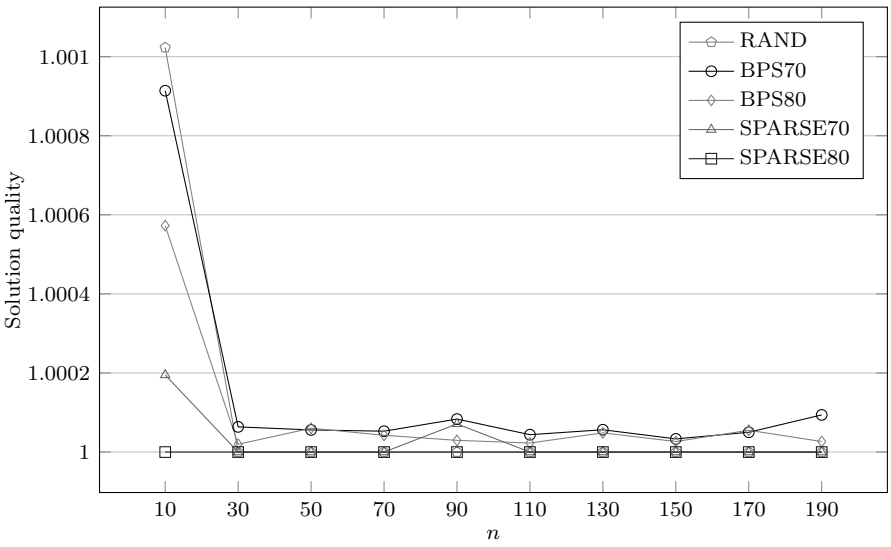

Figure 14: Quality of MP_{LS}.

et al. (2015). We have proposed a MILP formulation for PMMWM and provided a proof of the problem's strong NP-hardness. Moreover, we have presented two heuristic frameworks. For each framework, two variants have been designed. PM_{REG} and PM_{BPS} are based on the Partition-Match framework and MP and MP_{LS} are based on the Match-Partition framework. The performance of all four heuristics has been tested in an extensive computational study. We have generated instances of different types in order to create a comprehensive overview of the performance of the algorithms. Each of our heuristics has outperformed CPLEX. The Match-Partition framework has been more effective than the Partition-Match framework. Furthermore, MP_{LS} has shown to be robust against varying structures of the problem's underlying graph.

Acknowledgement

This work has been supported by the German Science Foundation (DFG) through the grant "Scheduling mechanisms for rail mounted gantries with regard to crane interdependencies" (PE 514/22-1).

References

Barketau, M., Pesch, E., & Shafransky, Y. (2015). Minimizing maximum weight of subsets of a maximum matching in a bipartite graph. *Discrete Applied Mathematics*, *in press*. doi:10.1016/j.dam.2015.01.008.

Boysen, N., & Fliedner, M. (2010). Determining crane areas in intermodal transshipment yards: The yard partition problem. *European Journal of Operational Research*, *204*, 336–342.

Boysen, N., Fliedner, M., Jaehn, F., & Pesch, E. (2013). A survey on container processing in railway yards. *Transportation Science*, *47*, 312–329.

Burkard, R., Dell'Amico, M., & Martello, S. (2009). *Assignment Problems*. Philadelphia: Siam.

Chen, B., Potts, C. N., & Woeginger, G. J. (1999). A review of machine scheduling: Complexity, algorithms and approximability. In D.-Z. Du, & P. M. Pardalos (Eds.), *Handbook of Combinatorial Optimization, Vol. 1* (pp. 1493–1641). Boston: Kluwer.

Domschke, W., & Scholl, A. (2005). *Grundlagen der Betriebswirtschaftslehre*. (3rd ed.). Berlin: Springer.

Garey, M. R., & Johnson, D. S. (1979). *Computers and Intractability – A Guide to the Theory of NP-Completeness*. New York: Freeman.

Isoloader (2012). When to choose a straddle and when to choose a forklift. URL: `http://www.isoloader.com/downloads/` [accessed online, June 2015].

Kemme, N. (2012). Effects of storage block layout and automated yard crane systems on the performance of seaport container terminals. *OR Spectrum*, *34*, 563–591.

Kozan, E., & Preston, P. (1999). Genetic algorithms to schedule container transfers at multimodal terminals. *International Transactions in Operational Research*, *6*, 311–329.

Kuhn, H. W. (1955). The Hungarian method for the assignment problem. *Naval Research Logistics Quarterly*, *2*, 83–97.

Lee, C.-Y., & Massey, J. D. (1988). Multiprocessor scheduling: Combining LPT and MULTIFIT. *Discrete Applied Mathematics*, *20*, 233–242.

Preston, P., & Kozan, E. (2001). An approach to determine storage locations of containers at seaport terminals. *Computers & Operations Research*, *28*, 983–995.