



Article

Multiple Hungarian Method for k -Assignment Problem

Boštjan Gabrovšek ^{1,2} , Tina Novak ¹, Janez Povh ^{1,3}, Darja Rupnik Poklucar ¹ and Janez Žerovnik ^{1,3,*} 

¹ Faculty of Mechanical Engineering, University of Ljubljana, Askerceva 6, SI-1000 Ljubljana, Slovenia; bostjan.gabrovsek@fs.uni-lj.si (B.G.); tina.novak@fs.uni-lj.si (T.N.); janez.povh@lecad.fs.uni-lj.si (J.P.); darja.rupnik@fs.uni-lj.si (D.R.P.)

² Faculty of Mechanical Engineering, University of Ljubljana, Jadranska ulica 19, SI-1000 Ljubljana, Slovenia

³ Institute of Mathematics, Physics and Mechanics, Jadranska 19, SI-1000 Ljubljana, Slovenia

* Correspondence: janez.zerovnik@fs.uni-lj.si

Received: 28 September 2020; Accepted: 13 November 2020; Published: 17 November 2020



Abstract: The k -assignment problem (or, the k -matching problem) on k -partite graphs is an NP-hard problem for $k \geq 3$. In this paper we introduce five new heuristics. Two algorithms, B_m and C_m , arise as natural improvements of Algorithm A_m from (He et al., in: Graph Algorithms And Applications 2, World Scientific, 2004). The other three algorithms, D_m , E_m , and F_m , incorporate randomization. Algorithm D_m can be considered as a greedy version of B_m , whereas E_m and F_m are versions of local search algorithm, specialized for the k -matching problem. The algorithms are implemented in Python and are run on three datasets. On the datasets available, all the algorithms clearly outperform Algorithm A_m in terms of solution quality. On the first dataset with known optimal values the average relative error ranges from 1.47% over optimum (algorithm A_m) to 0.08% over optimum (algorithm E_m). On the second dataset with known optimal values the average relative error ranges from 4.41% over optimum (algorithm A_m) to 0.45% over optimum (algorithm F_m). Better quality of solutions demands higher computation times, thus the new algorithms provide a good compromise between quality of solutions and computation time.

Keywords: k -assignment problem; k -matching problem; heuristic algorithm; local search; greedy algorithm; hungarian method

1. Introduction

1.1. Motivation

Suppose we have k sets of vertices V_1, V_2, \dots, V_k and we want to consider multi-associations between them. For example, in bioinformatics, V_1 can correspond to the set of known (relevant) diseases, V_2 to the set of known drugs, and V_3 to the set of known genes that are relevant for the observed species (e.g., for Homo Sapiens). Multi-association in this case is a triple $(v_1, v_2, v_3) \in V_1 \times V_2 \times V_3$, which means that disease v_1 , drug v_2 , and gene v_3 are related. Such a triple may imply that the gene v_3 is activated in disease v_1 and is usually silenced by drug v_2 , hence drug v_2 may be considered to be the cure for disease v_1 . This is related to the very vibrant area of drug re-purposing and precision medicine, see e.g., [1–3]. We can represent the data as a complete 3-partite graph where the vertex set is $V_1 \cup V_2 \cup V_3$ and the edges between vertices from different V_i have weights equal to the strength of the association between the ending vertices. Each triple (v_1, v_2, v_3) is therefore a complete subgraph (3-clique, triangle) of such a graph and its weight is the sum of the weights on its edges. If we want to find a decomposition of this graph into disjoint triangles with a maximum (minimum)

total weight, we obtain the 3-assignment problem (3-AP) [4–7]. The 3-AP can also serve as a model in production planning when we try to assign e.g., workers, machines, and tasks in a way that each worker gets exactly one task at one machine and the total cost is minimal. Many more applications can be found in the literature, see cf. [8] or [9], and the references therein.

1.2. Problem Formulation

Let $G = (V, E, w)$ be a complete weighted k -partite graph where $V = V_1 \cup V_2 \cup \dots \cup V_k$ is the vertex set, V_i are the vertices of the i -th partition with cardinality $|V_i| = n$, $E = \bigcup_{1 \leq i < j \leq k} \{uv \mid u \in V_i, v \in V_j\}$ is the edge set, and $w : E \rightarrow \mathbb{R}$ is the weight function that may be given in terms of matrices W^{ij} as $w(e) = W_{uv}^{ij}$ for $e = uv$, $u \in V_i$, $v \in V_j$. A k -clique is a subset $Q \subset V$ with cardinality k , such that the induced graph $G[Q]$ is isomorphic to the complete graph K_k . This means that a k -clique has exactly one vertex from each V_i . In the case when G is a k -partite graph, a k -clique can be also called a k -association. The weight of a k -clique Q , $w(Q)$, is the sum of the edge weights of $G[Q]$:

$$w(Q) = \sum_{e \in E(G[Q])} w(e).$$

In a complete k -partite graph where each partition has cardinality n , we can always find n pairwise disjoint k -cliques $\mathcal{Q} = \{Q_1, Q_2, \dots, Q_n\}$. We call such a set of cliques a k -assignment or k -matching, since this is a natural extension of 2-assignments or 2-matchings in the bipartite graphs. Naturally, we define the weight of k -assignment \mathcal{Q} as

$$w(\mathcal{Q}) = \sum_{i=1}^n w(Q_i).$$

The k -assignment problem, or equivalently, the k -matching problem (k-AP) (1), is the problem of finding a k -assignment of a given k -partite graph G with the minimum weight:

$$\min\left\{\sum_{\mathcal{Q}} w(\mathcal{Q}) \mid \mathcal{Q} \text{ is a } k\text{-assignment in } G\right\}. \quad (1)$$

In the literature [4–6,10], this problem is also referred to as the multidimensional assignment problem (MAP). For the case $k = 3$ we can also trace the name 3-index assignment problem or 3-dimensional assignment problem in the literature. When $k = 2$, it is well-known that the Hungarian algorithm solves the 2-assignment problem to optimality [11] in polynomial time. Kuhn used this name for the method because his invention of the algorithm is based on the work of two Hungarian mathematicians, D. König and E. Egervary. We observe that sometimes the researchers use word matching if the weights on the graph edges are all equal to 1, while for the general case they use assignment.

In this paper, we will also consider the maximum version of (1) because we want to compare our heuristic algorithms with some algorithms from the literature correctly. To make a clear distinction, we use subscripts m and M in the names of heuristic algorithms to denote that we are solving (1) with minimum and maximum objective, respectively.

We conclude the subsection with a useful observation. For $k > 2$, every k -assignment \mathcal{Q} implies a 2-assignment on $G[V_i \cup V_j]$, for all $i \neq j$. Thus \mathcal{Q} gives rise to $\binom{k}{2}$ 2-assignments $M_{i,j}$, $1 \leq i < j \leq k$, between partitions V_i and V_j . Therefore, a k -assignment \mathcal{Q} defines $\binom{k}{\ell}$ ℓ -assignments on subgraphs of G induced on $\binom{k}{\ell}$ different ℓ -partitions.

1.3. Literature Review

The problem has been extensively studied in the past. Here we briefly mention some of the relevant results and cite some previous work without the intention to review the literature completely. The problem called 3-dimensional matching (3DM) has already appeared among the

NP-hard problems in Karp's [12] seminal paper. This problem is related to the question of whether there exists a 3-assignment in a 3-partite graph if the partitions of the graph have the same cardinality but the graph is not necessary a complete k -partite graph.

According to [8], 3DM is a special case of the 3-assignment problem with maximum or minimum objective, which they call the axial 3-index assignment problem, hence Karp's result [12] implies that both minimization and maximization versions of 3-assignment problems are NP-hard.

If $k = 3$ and if we consider only the weights on the triples Q_i which must be 0 or 1, then there exists a $(\frac{2}{3} - \epsilon)$ -approximation algorithm [13]. If the weights on the triangles are arbitrary, there exists a $(\frac{1}{2} - \epsilon)$ -approximation algorithm [14].

For the minimization version of the problem, it is known [15] that there is no polynomial time algorithm that achieves a constant performance ratio unless $P=NP$ and the result holds even in the case when the clique weights are of the form

$$w_{ijk} = d_{ij} + d_{ik} + d_{jk}$$

for all i, j, k (i.e., for the problem defined here as (1)). However, when the triangle inequality holds, Crama and Spieksma show that there is a $\frac{4}{3}$ -approximation algorithm [15].

Hence, it is justified to apply heuristics to find near optimal solutions of the 3-AP and in general for k -AP problem instances. In the literature, various heuristics are reported that were designed to handle the k -assignment problem, many focusing on the 3-AP. We mention some of them to illustrate the variety of ideas elaborated. Aiex et al. [16] adopted the heuristic called *Greedy Randomized Adaptive Search Algorithm*. Huang and Lim in [17] described a new local search procedure which solves the problem by simplifying it to the classical assignment problem. Furthermore, Huang and Lim hybridized their heuristic with a *genetic algorithm*. An extension of the Huang and Lim heuristic [17] to the multidimensional assignment problem was done in [18,19], while [20] developed another new heuristic named *Approximate muscle guided beam search*, using the local search of [17]. Karapetyan and Gutin [21] devised a *memetic algorithms* with an adjustable population size technique and with the same local search as Huang and Lim for the case of 3-AP. The size was calculated as a function of the runtime of the whole algorithm and the average runtime of the local search for the given instance. According to Valencia, Martinez, and Perez [22] this is the best known technique to solve the general case of k -AP. These authors performed an experimental evaluation of a basic *genetic algorithm* combined with a *dimensionwise variation heuristic* and showed its effectiveness in comparison to a more complex state-of-the-art memetic algorithm for k -AP. Some other approaches were recently proposed for solving 3-AP [23], the so-called *Neighborly algorithm*, *modified Greedy algorithm* with some of the steps used by the *Auction algorithm* [24], and a probabilistic modification of the *minimal element algorithm* for solving the axial three-index assignment problem [25], where the idea was to extend the basic greedy-type algorithmic schemes using transition to a probabilistic setup based on variables randomization.

The k -assignment problem can be also formulated as an integer (0–1) linear programming problem in n^k binary variables. This approach yields some interesting theoretical results, but has very limited practical impact due to the huge number of binary variables. More details can be found in [5,26]. Some recent results related to special variants of the k -assignment problem can also be found in [23,27].

An exact algorithm for 3-AP is proposed by Balas and Saltzman [4]. For more information on related work we refer to [8,9] and the references there.

The idea to use the Hungarian algorithm for 2-AP as a tool to attack the k -AP first appears in [28], where the algorithm named A_m (see the descriptions in the next section) for the approximate solution to the minimal k -assignment problem and an algorithm A_M for the approximate solution to the maximal k -assignment problem (or the maximal k -clique problem) of a weighted complete k -partite graph is given. In [28] it is experimentally shown that the *Cubic Greedy Algorithms* are better than the *Random Select Algorithm* and that Algorithms A_m and A_M are better than the *Cubic Greedy Algorithms*. For $k = 4$, it is also shown that Algorithms A_m and A_M are better than the *4-clique Greedy Algorithm*.

1.4. Our Contribution

As the (1) problem is NP-hard for $k \geq 3$, it is natural to ask whether one can design a useful heuristic based on the Hungarian algorithm that efficiently solves the $k = 2$ case to optimality. The first work along this avenue is the implementation of Algorithm A from [28]. We continue the research with the main goal to understand how much the ideas of the Hungarian algorithm can contribute to performance of the heuristics. To this aim, we design several heuristics that are based on the Hungarian algorithm. Our experimental results show that all the algorithms improve the quality of the solutions compared to A, and some of our algorithms are a substantial improvement over the basic algorithm A (see Table 2). We also show that this type of heuristics can provide near optimal solutions for the k -assignment problem of very good quality on the datasets with known optimal values (see Tables 3 and 4).

The experiments are run on two datasets from the literature, one of them providing optimal solutions for the instances. In addition, we run two batches of experiments including instances generated by nonuniform distribution, in contrast to other datasets where the uniform distribution is used. Due to intractability of the k -assignment problem, it is not a surprise that our experimental study shows limitations of particular heuristics. Therefore we also introduce two randomized versions of heuristic algorithm C that lead to local search type heuristics that are observed to be improving the quality of solutions over time and may converge to the optimal solution. In this way we complement the quick heuristics with some alternatives that trade much longer computational times for quality of solutions. Hence we show that the heuristics for the k -assignment problem that are based on the Hungarian algorithm may be a very competitive choice.

The main contributions of this paper are the following:

1. We design five new heuristics for (1). More precisely, we propose algorithms named B, C, D, E, and F for finding near optimal solutions to the (1). (All the algorithms have both the minimization and maximization versions, that are respectively denoted c.f. X_m and X_M , for algorithm X.) The algorithms rely on heavy usage of the Hungarian algorithm [11] and arise as natural improvements of Algorithm A_m from [28]. The last two algorithms, E and F, can be considered as versions of iterative improvement type local search algorithms, as opposed to the steepest descent nature of, e.g., C.
2. We implement and test the algorithms on three datasets:
 - (a) The set of random graphs generated as suggested in [28]. Here we also reproduce their results.
 - (b) We design two random datasets using both the uniform and a nonuniform distribution for the second batch of experiments.
 - (c) We test our algorithms on the dataset of hard instances from [15], for which the optimal solutions are known.
3. Our experimental results show that (on all datasets used) all the algorithms improve the quality of the solutions compared to algorithm A. We also observe the algorithms performance considering the quality of solutions versus computation time.

The rest of the paper is organized as follows. In Section 2 we introduce the notation that is used throughout the paper, and in Section 3 we outline the algorithms. In Section 4 we provide results of our experiments, where we evaluate the existing and the new algorithms on the three datasets. In Section 5 we summarize the results and discuss the methods of choice.

2. Preliminaries

From k -Assignment Problem to 2-Assignment Problem

The Hungarian method [11] is a classical and polynomial-time exact method for solving the 2-assignment problem, therefore it is very natural that we explore the idea to find near optimal feasible solutions of k -assignment problem by solving a series of 2-assignment problems.

Below we are going to present several new algorithms for (1), which strongly rely on the repetitive use of the Hungarian method on selected bipartite subgraphs and contracting the original graphs along the assignments computed by this method. Note that all algorithms return feasible solutions because we have a complete k -partite graph where the Hungarian method always finds an optimum solution for any induced graph $G[V_i \cup V_j]$, which can be easily used to reconstruct a feasible solution of (1) via operator $*$, see Section 3. Given an ℓ -partite weighted graph G and a 2-assignment $M_{i,j}$ between partitions V_i and V_j of G , we wish to contract the (weighted) edges which constitute $M_{i,j}$ in G . We therefore associate to $M_{i,j}$ the quotient graph $G/M_{i,j} = G/\sim$, where $u \sim v \Leftrightarrow uv \in M_{i,j}$. The construction is explained in more detail below. The new weight function of $G/M_{i,j}$ is obtained by summing the weights of contracted edges adjacent to $M_{i,j}$. Formally, the vertices of $G/M_{i,j}$ are the equivalence classes consisting of either the singleton $\{v\}$ if the vertex v is not adjacent to an edge in the assignment $M_{i,j}$ or $\{u, v\}$ if uv is an edge in $M_{i,j}$. Loosely speaking, in $G/M_{i,j}$, pairs of elements of V_i and V_j are merged along $M_{i,j}$ to form a new partition U' while the other $V_k, k \neq i, k \neq j$ are not changed. (See Figure 1a,b.) More formally,

$$U' = \{\{u, v\} \mid uv \in M_{i,j}\} \quad \text{and} \quad V'_\ell = \{\{v\} \mid v \in V_\ell\}.$$

By construction, the elements of U' and V'_ℓ are equivalence classes, and formally we have $\{u, v\} = [u] = [v]$ and $\{u\} = [u]$. However, we will (warning that it is abuse of notation) often not distinguish between V'_ℓ and V_ℓ , (i.e., identify $V'_\ell = V_\ell$) and also consider the elements of U' as sets of two elements, the union of two singleton sets. Formally, $G/M_{i,j} = (V', E', w')$, is the graph with the vertex set

$$V' = U' \cup \bigcup_{\substack{1 \leq l \leq k \\ l \neq i, l \neq j}} V'_l,$$

and the edge set

$$E' = F' \cup \bigcup_{\substack{1 \leq i' < j' \leq k \\ \{i', j'\} \cap \{i, j\} = \emptyset}} E_{i', j'}$$

where $F' = \{\{u\}\{v, v'\} \mid u \in V \setminus (V_i \cup V_j), vv' \in M_{i,j}\}$ and $E_{i', j'} = \{\{u\}\{v\} \mid uv \in E \setminus M_{i,j}\}$. Or, recalling the simplified notation above, simply $E_{i', j'} = E_{i, j}$.

The new weight function is defined by summing the weights on the edges adjacent to the identified vertices: if $u, v \in V \setminus (V_i \cup V_j)$, then we have

$$w'(\{u\}\{v\}) = w(uv)$$

and if $u \in V \setminus (V_i \cup V_j)$ and $vv' \in M_{i,j}$, we have

$$w'(\{u\}\{v, v'\}) = w(uv) + w(uv').$$

In other words, the weights on $E_{i', j'} = E_{i, j}$ are not changed, and the weights on the edge set F' are the weights of pairs of edges that were contracted to obtain triangles $\{u, v, v'\}$.

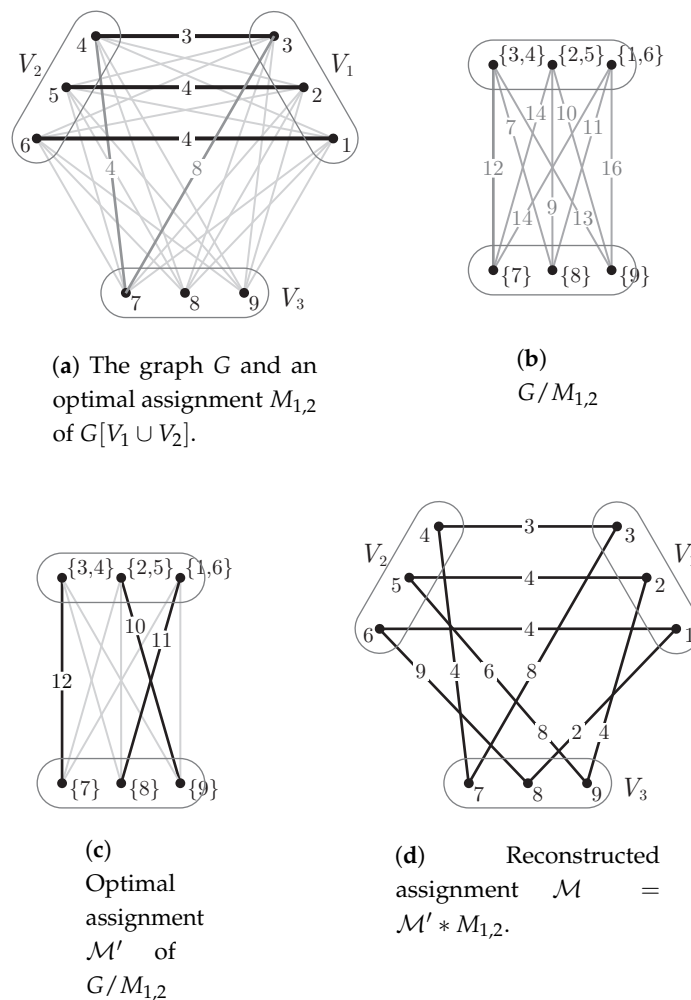


Figure 1. Application of Algorithm A_m on a small example.

3. Algorithms

In this section we first recall Algorithm A_m for (1) from [28] and then enhance it using different well-known greedy and local search approaches. Let A be a set and f a function $f : A \rightarrow \mathbb{R}$. Denote by

$$\arg \min_{x \in A} f(x) = \{x \mid \forall y \in A : f(y) \geq f(x)\},$$

the set of minimal elements in A under the function f . Let $M_{i,j}$ be an arbitrary assignment between the i -th and j -th partition and let \mathcal{M}' be a $(k-1)$ -assignment for the $(k-1)$ -partite graph $G/M_{i,j}$. We denote by $\mathcal{M}' * M_{i,j}$ the unique k -assignment for G reconstructed from \mathcal{M}' and $M_{i,j}$, i.e.,

$$\mathcal{M}' * M_{i,j} = \{uv \mid [u][v] \in \mathcal{M}'\} \cup \left(\bigcup_{\substack{[u][v'] \in \mathcal{M}' \\ v' \in M_{i,j}}} \{uv, uv'\} \right).$$

For an example see Figure 1. In the case when G is a bipartite graph, $G/M_{i,j}$ contains only one partition and thus does not allow any assignment, for completeness we thus define $\emptyset * M_{i,j} = M_{i,j}$. Recall that in the case $n = 2$ an optimal 2-assignment can be found using the Hungarian algorithm [11]. The result of this algorithm on bipartite graph G will be denoted by $\text{HUNGARIAN}(G)$.

Algorithm 1, A_m . The following algorithm, which we denote by A_m , for finding near optimal solution for the minimal k -assignment problem of k -partite graph G has been proposed in [28].

Algorithm 1 [28].

```

1: function  $A_m(G)$ 
2:   if  $k = 1$  then
3:     return  $\emptyset$ 
4:   else
5:      $M_{1,2} = \text{HUNGARIAN}(G[V_1 \cup V_2])$ 
6:     return  $A_m(G/M_{1,2}) * M_{1,2}$ 

```

In short, the algorithm finds an optimal 2-assignment for $G[V_1 \cup V_2]$, takes the quotient by this assignment and recursively repeats this process. The final result is a complete k -assignment \mathcal{M} reconstructed from the previously computed $(k - 1)$ -assignments.

Example 1. Let G be a complete 3-partite graph with partitions $V = V_1 \cup V_2 \cup V_3$, $V_1 = \{1, 2, 3\}$, $V_2 = \{4, 5, 6\}$, $V_3 = \{7, 8, 9\}$ and the following (weighted) adjacency matrix

$$W = \begin{bmatrix} 0 & 0 & 0 & 9 & 6 & 4 & 8 & 2 & 9 \\ 0 & 0 & 0 & 4 & 4 & 6 & 5 & 0 & 4 \\ 0 & 0 & 0 & 3 & 9 & 2 & 8 & 7 & 4 \\ \hline 9 & 4 & 3 & 0 & 0 & 0 & 4 & 0 & 9 \\ 6 & 4 & 9 & 0 & 0 & 0 & 9 & 9 & 6 \\ 4 & 6 & 2 & 0 & 0 & 0 & 8 & 9 & 5 \\ \hline 8 & 5 & 8 & 4 & 9 & 8 & 0 & 0 & 0 \\ 2 & 0 & 7 & 0 & 9 & 9 & 0 & 0 & 0 \\ 9 & 4 & 4 & 9 & 6 & 5 & 0 & 0 & 0 \end{bmatrix},$$

where the entries generate the weight function $w : uv \mapsto W_{u,v}$; $u, v \in \{1, \dots, 9\}$.

In Algorithm A_m we first compute the optimal assignment $M_{1,2}$ between partitions V_1 and V_2 (Figure 1a), then we compute the quotient graph $G/M_{1,2}$, presented in Figure 1b, and finally we compute the optimal assignment for this graph (Figure 1c). At the end, we reconstruct the 3-assignment \mathcal{M} and obtain a solution of weight 44 (Figure 1d).

Algorithm 2, B_m . Algorithm A_m is greedy in the sense that it takes (lexicographically) the first pair of partitions and merges them according to the best assignment among them. If the order of partitions is changed, A_m would provide a different result. The idea of B_m is to consider all the pairs for possible first merge, and take the best result. Note that B_m is also greedy as it always takes the minimal assignment between the two partitions that are merged. (In a sense, one may say that B_m is somehow more greedy than A_m because it looks a bit farther for the seemingly best option in sight.)

Algorithm 2 (B_m , improvement of A_m).

```

1: function  $B_m(G)$ 
2:   if  $k = 1$  then
3:     return  $\emptyset$ 
4:   else
5:      $\{a, b\} \in \arg \min_{1 \leq i < j \leq k} w(B_m(G/\text{HUNGARIAN}(G[V_i \cup V_j])))$ 
6:     return  $B_m(G/M_{a,b}) * M_{a,b}$ 

```

Algorithm B_m searches through all possible pairs (V_i, V_j) of partitions, recursively runs on the quotient graph G/M , where M is the optimal 2-assignment on $V_i \cup V_j$, and among these partitions

chooses the one with the best assignment of G/M . If there are more minimal partitions, the algorithm chooses a random partition of minimal weight. Clearly, Algorithm B_m returns a k -assignment that is determined by the $(k - 1)$ 2-assignments that were chosen in the recursive calls of B_m .

Example 2. We take the same graph as in Example 1. In contrast to A_m , Algorithm B_m finds optimal assignments $M_{1,2}$, $M_{1,3}$, and $M_{2,3}$ for the induced subgraphs $G[V_1 \cup V_2]$, $G[V_1 \cup V_3]$, and $G[V_2 \cup V_3]$, see Figure 2a–c, respectively. The algorithm continues its search recursively on the bipartite graphs $G/M_{1,2}$, $G/M_{1,3}$, and $G/M_{2,3}$. As Figure 2b shows, we obtain a k -assignment of weight 40.

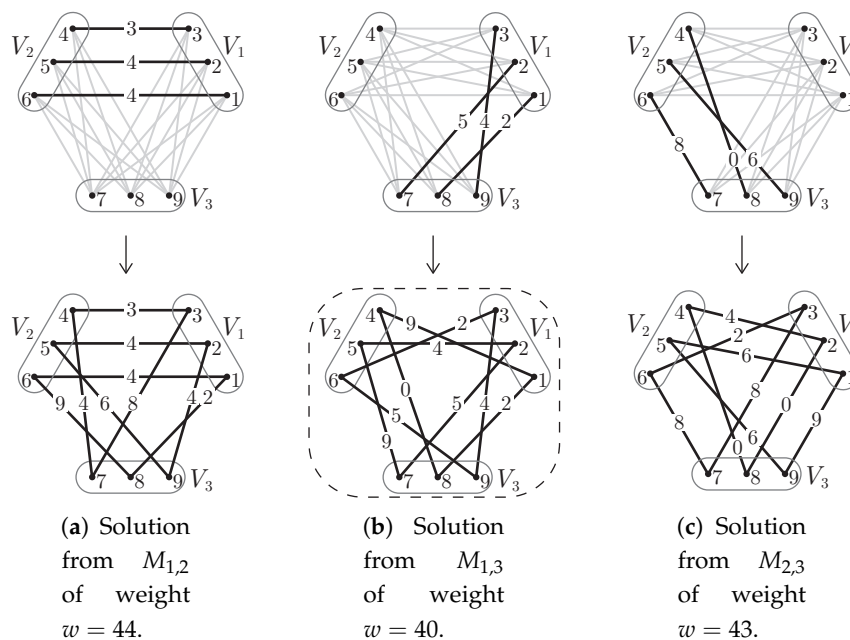


Figure 2. Cases for Algorithm B_M .

Algorithm 3, C_m . Observe that Algorithm B_m is much more time consuming than Algorithm A_m as it calls the Hungarian algorithm subroutine

$$\binom{k}{2} \binom{k-1}{2} \cdots \binom{3}{2} = \frac{k!(k-1)!}{2^{k-1}}$$

times as opposed to only $(k - 1)$ calls by Algorithm A_m .

However, note that Algorithm B_m is greedy because it always takes the minimal 2-assignment. As the k -assignment problem (for $k > 2$) is intractable, a deterministic greedy algorithm can not solve the problem to optimality unless $P=NP$. We therefore consider an iterative improvement of the solutions by taking a nearly optimal solution (that may be the result of B_m) to define an initial solution. The neighbors of the given solution are the results of the following procedure: fix one of the 2-assignments, say M , and run Algorithm B_m on G/M . After repeating the process by fixing all 2-assignments, we get a set of new solutions. If at least one of them is an improvement over the previously known best solution, we continue the improvement process. The process stops when there is no better solution among the set of new solutions.

The third algorithm, denoted as Algorithm C_m , can be considered as a steepest descent algorithm on the set of all k -assignments, where the next solution is chosen to be a minimal solution in a suitably defined neighborhood.

Algorithm 3 (C_m , steepest descent based on B_m).

```

1: function  $C_m(G)$ 
2:    $\mathcal{M} = B_m(G)$ 
3:   repeat
4:      $\mathcal{M}_{\text{previous}} = \mathcal{M}$ 
5:      $M_{a,b} = \arg \min_{M_{i,j} \in \mathcal{M}} w(B_m(G/M_{i,j}))$ 
6:      $\mathcal{M} = B_m(G/M_{a,b}) * M_{a,b}$ 
7:   until  $w(\mathcal{M}_{\text{previous}}) = w(\mathcal{M})$ 
8:   return  $\mathcal{M}$ 

```

Example 3. Taking the same instance as in Examples 1 and 2, Algorithm C_m starts with finding a 3-assignment $\mathcal{M} = \{M_{1,2}, M_{1,3}, M_{2,3}\}$ using B_m and continues by recursively searching graphs $G/M_{1,2}$ and $G/M_{2,3}$ (we can skip $G/M_{1,3}$, since the initial 3-assignment was already obtained from $G/M_{1,3}$). As Figure 3 shows, the best solution of weight 37 is obtained from contracting $M_{2,3}$. If we continue with the iteration, we can see that the solution has stabilized and no further improvements can be made.

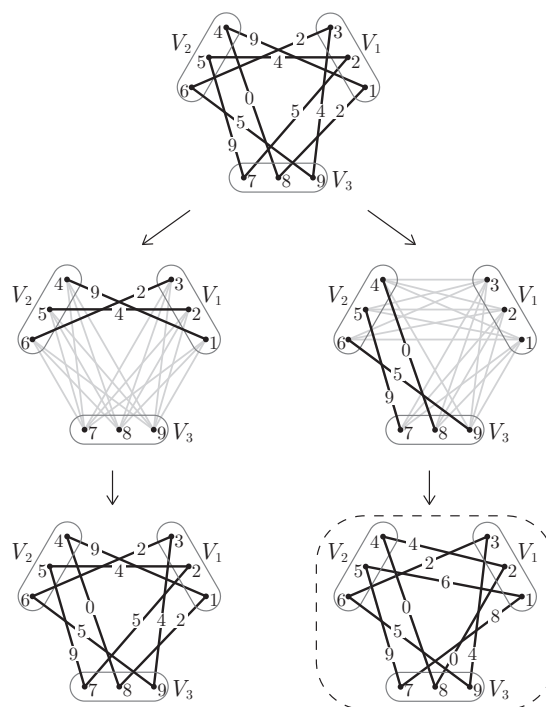


Figure 3. Cases for Algorithm C_m .

Algorithm 4, D_m . Algorithms B_m and C_m heavily rely on the Hungarian method and are very time-consuming in comparison to A_m . Therefore we define another greedy algorithm based on the Hungarian method that is faster. We denote by D_m the greedy algorithm that takes the minimal 2-assignment $M_{i,j}$ in the k -partite graph G , and continues considering the $(k - 1)$ -partite graph $G/M_{i,j}$ until only one partition is left.

Algorithm 4 (D_m , Greedy iterative)

```

1: function  $D_m(G)$ 
2:   if  $k = 1$  then
3:     return  $\emptyset$ 
4:   else
5:      $\{a, b\} \in \arg \min_{1 \leq i < j \leq k} w(\text{HUNGARIAN}(G[V_i \cup V_j]))$ 
6:     return  $D_m(G/M_{a,b}) * M_{a,b}$ 

```

Note that Algorithm D_m has the shortest average running time among algorithms B, C, and D. On the other hand, it is at the same time also the most short-sighted greedy algorithm among the rest of the algorithms, since its choice is based on the quality of the 2-assignment at hand, $w(M)$, and not by looking at the quality of $w(G/M)$.

Algorithm 5, E_m . The algorithms considered above were in principle deterministic. Except breaking ties randomly in case of more than one minimal choice, all the steps are precisely defined, i.e., the algorithms are deterministic. In the final experiment, we are interested in the effect of randomization.

The next algorithm, E_m , is an alternative randomized version of C_m . The main idea is to loop through all 2-assignments in random order and accept the first assignment $M_{i,j}$ that yields a better solution, instead of searching for the minimal 2-assignment $M_{i,j}$. Thus algorithm E_m is obtained by changing the main loop of C_m . As algorithm E_m accepts the first better solution in the neighborhood of the current solution, it is a kind of iterative improvement algorithm as opposed to C_m , which is a steepest descent type local search algorithms, because it always looks for the best solution in the neighborhood.

Algorithm 5 (E_m)

```

1: function  $E_m(G)$ 
2:    $\mathcal{M} = B_m(G)$ 
3:   repeat
4:      $min\_weight = w(\mathcal{M})$ 
5:     for  $(i, j) \in \text{SHUFFLE}(\binom{k}{2})$  do
6:        $\mathcal{M} = \min\{\mathcal{M}, B_m(G/M_{i,j}) * M_{i,j}\}$ 
7:       if  $min\_weight \neq w(\mathcal{M})$  then
8:         break
9:   until  $min\_weight = w(\mathcal{M})$ 
10:  return  $\mathcal{M}$ 

```

We denote by $E_m(n)$ the algorithm that takes the best solution of E_m out of n trials (restarts).

Algorithm 6, F_m . The Algorithm E_m stops when there is no better solution, even if there are solutions of the same quality (weight) in the neighbourhood of the current solution. These neighborhood solutions might later give improvement, therefore we introduce another variant, called $F_m(n)$, which in such case chooses randomly one solution of equal weight and continues, but stops after at most n steps, since it does not necessarily terminate (e.g., it can iterate between two equally good solutions).

Algorithm 6 (F_m , multistart local search)

```

1: function  $F_m(G, n)$ 
2:    $\mathcal{M} = B_m(G)$ 
3:    $last\_assignment = \text{none}$ 
4:   for  $counter = 1, \dots, n$  do
5:      $assignments = \arg \min_{M_{i,j} \in \mathcal{M}} w(B_m(G/M_{i,j}) * M_{i,j})$ 
6:     if  $last\_assignment$  in  $assignments$  then
7:       remove  $last\_assignment$  from  $assignments$ 
8:     if  $assignments = \emptyset$  then
9:       break
10:     $M_{a,b} = \text{CHOOSE\_RANDOM}(assignments)$ 
11:     $\mathcal{M} = B_m(G/M_{a,b}) * M_{a,b}$ 
12:  return  $\mathcal{M}$ 

```

All of the above algorithms can be easily adapted to solve maximization assignment problems, i.e., to solve (1) where the objective is maximum. We denote the maximization variants of the algorithms A, B, C, D, E and F by A_M , B_M , C_M , D_M , E_M , and F_M , respectively.

Remark 1. Clearly, the algorithms C, D, E, and F always return a feasible assignment because any solution is obtained by a recursive call of B. However, many calls of B and thus many runs of the Hungarian algorithm are expensive in terms of computation time. Therefore, it is an interesting question whether the present local search heuristics may be sped up by considering other neighborhoods, for example applying the idea of variable neighborhood search [29].

4. Numerical Results

In this section, we present numerical evaluations of the algorithms introduced in Section 3. We compare them with the algorithms from [28] and to each other. In particular, we

- reproduce the results on random graphs as given in [28] and compare them with the results of our Algorithms B_m to F_m and their maximization variants B_M to F_M ,
- evaluate the performance of the algorithms against A_M as the number of vertices increases,
- test our algorithms on the instances provided in [15].

4.1. Datasets

Numerical evaluations are done using three sets of random complete k -partite graphs. The first set was constructed according to [28], as follows: It consists of two sets of 1000 random complete k -partite graphs with $k = 3, 4$ and $n = 30, 100$, respectively. The weights on the edges were selected randomly from given set S with probability density function $p(x) = \frac{1}{|S|}, \forall x \in S$, where $S = \{0, 1, \dots, 9\}$, if $k = 3$ and $S = \{1, \dots, 100\}$, if $k = 4$.

The second dataset is our contribution. It has been designed to compare how our algorithms scale with increasing size of the instances. It consists of two subsets, each consisting of instances with $k = 3$ and $k = 4$. The first subset was generated as follows. For each $k \in \{3, 4\}$ we range the number of vertices n in each partition from 2 to 100 and the weights on edges connecting vertices from different partitions are chosen randomly according to discrete uniform distribution on the set $S = \{0, \dots, n-1\}$. The second subset was obtained similarly, we only changed the distribution of the edge weights. The edges between the different partitions are assigned random weights chosen according to discrete uniform distribution on the set $S = \{2^0, 2^1, \dots, 2^{10}\}$. We expect that these random instances are more difficult, because the very important edges are sufficiently rare. For each pair k, n we generated 1000 random instances.

The third set is the same as in [15]. For this set, the optimum value of 3-AP is known. We retrieved it from [30]. This dataset includes 18 instances of complete 3-partite graphs:

- 6 graphs with 33 vertices and 6 graphs with 66 vertices in each of the 3 partitions, where the weights of edges between different partitions are random integers which should, according to the description given by the authors [30], range from 0 to 99. However, we point out that some of the weights in these instances are larger than 100.
- 3 graphs with 33 vertices and 3 graphs with 66 vertices in each partition, where the weights take only values 1 or 2. We call these graphs binary graphs.

At the beginning of the web page [30] it is explained how the numerical results from [15] relate to these instances.

4.2. First Experiment—Dataset from He et al. (2004)

In the first experiment we compare the algorithms used in [28] (namely, the Random, Greedy, and A_M algorithm) with our Algorithms $B_M, C_M, D_M, E_M, E_M(10)$, and $F_M(100)$ on the first set of random k -partite graphs, which we generated as described in [28], see Section 4.1. We run these 5 algorithms on each group of 1000 graph instances and report the average values in Tables 1 and 2.

Table 1. Comparison of Random, Greedy and A_M algorithms from [28] with B_M to F_M algorithms for the maximization version of 3-AP. Each row contains average values of solutions obtained by these algorithms, computed over 1000 random instances of complete k -partite graphs with n vertices, which are generated as described in Section 4.1. We can see that Algorithms B_M to F_M return substantially better results.

Algorithm	$k = 3, n = 30,$ $S = \{0, \dots, 9\}$			$k = 4, n = 100,$ $S = \{1, \dots, 100\}$		
	val	$\Delta_{A_M}(\%)$	Time	val	$\Delta_{A_M}(\%)$	Time
Random	405	−45.9	-	41806	−23.5	-
Greedy	736	−1.8	-	52801	−3.0	-
A_M	749.2	0	1	54,421.7	0	1
B_M	753.8	0.6	3.0	54,634.1	0.4	14.4
C_M	759.4	1.4	6.1	54,731.5	0.6	46.7
D_M	749.4	0.0	2.4	54,442.9	0.0	3.7
E_M	759.3	1.3	5.3	54,730.5	0.6	37.9
$E_M(10)$	759.9	1.4	53.0	54,761.1	0.6	378.7
$F_M(100)$	760.4	1.5	94.8	54,732.0	0.6	674.7

Table 2. The rows contain (respectively) the average values obtained by algorithms Random, Greedy, A_m from [28], and by our Algorithms B_m to C_m over 1000 random complete 3-partite graphs, respectively. We can see that our Algorithms B_m to F_m outperform the algorithms from [28].

Algorithm	$k = 3, n = 30,$ $S = \{0, \dots, 9\}$		
	val	$\Delta_{A_m}(\%)$	Time
Random	405	566.7	-
Greedy	218	258.9	-
A_m	60.7	0	1
B_m	56.2	−7.4	3
C_m	50.8	−16.4	6.1
D_m	60.6	−0.2	2.4
E_m	50.9	−16.2	5.3
$E_m(10)$	50.3	−17.2	53
$F_m(100)$	49.8	−18.0	95.2

In order to compare the algorithms with A_m (resp. A_M), we define the relative gap with respect to the value obtained by A_m (resp. A_M) by

$$\Delta_{A_m} = 100 \cdot \frac{\text{val} - \text{val}_{A_m}}{\text{val}_{A_m}} \quad \left(\text{resp. } \Delta_{A_M} = 100 \cdot \frac{\text{val} - \text{val}_{A_M}}{\text{val}_{A_M}} \right).$$

4.3. Second Experiment on Random Instances

In this subsection we compare Algorithms B_M , C_M , D_M , E_M , $E_M(10)$, and $F_M(100)$ on the second dataset, introduced in Section 4.1. We run all three algorithms on each of two subsets, consisting of 1000 instances for each pair $(k, n) \in \{3, 4\} \times \{2, 3, 4, \dots, 100\}$. For each pair (k, n) and each algorithm, we compute the average value of solutions given by the algorithm over the corresponding 1000 instances. Then, we compute quotients of the average values for B_M and for A_M and denote it by B_M/A_M . Similarly we compute quotient C_M/A_M , and so on. Figures 4–7 contain plots and interpretations of these quotients.

The results on the first subset (with uniform distribution of weights) are depicted on Figures 4 and 5. They show that Algorithm $E_M(10)$ clearly finds the best solutions. The Algorithms C_M , E_M and $F_M(100)$ perform similarly, and are clearly outperforming B_M and D_M . Note that taking into account time complexity and considering $k = 3$, the clear winners among the faster algorithms (A_M , B_M , C_M , D_M , and E_M) are C_M and E_M , and among the more time consuming $E_M(10)$ and $F_M(100)$,

the winner is $E_M(10)$. Note that $E_M(n)$ may potentially find even better solutions with larger n (and consequently may need more time). The differences are much less obvious for $k = 4$ (see Figure 5). With larger n , the ratios seem to stabilize at certain constants.

Considering the results on the first dataset (Table 1) and the first sample of the second dataset (Figures 4 and 5) suggest that there is no significant difference in quality of solutions among the algorithms C, E, and F. However, the results on the second subset (the set with a special distribution of weights), in particular for $k = 3$, show that Algorithms C, E, and F substantially outperform B and D (see Figure 6), and the differences of ratios tend to grow with larger n . This allows us to conclude that Algorithms C_M , $E_M(n)$ and $F_M(n)$ are significantly better than B_M and D_M (at least on most of our instances). For $k = 4$ (see Figure 7), the differences are small again.

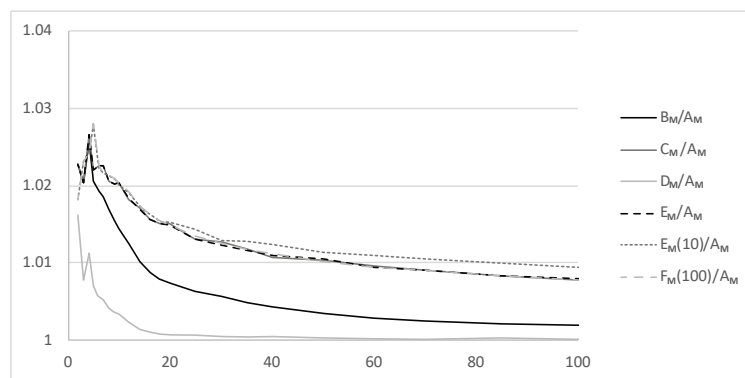


Figure 4. This plot depicts the quotients with A_M for the instances from the first subset of the third dataset (see Section 4.1) corresponding to $k = 3$. The x axis represents the size of each partition n , while the y axis represents the quotient. We can see that with larger n , $E_M(10)$ outperforms all other algorithms, while C_M , E_M , and $F_M(100)$ perform similarly.

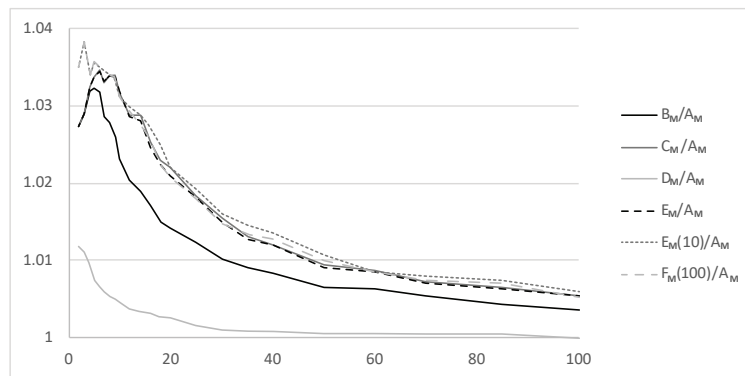


Figure 5. On this plot we can observe the quotients with A_M for the instances from the first subset of the third dataset, corresponding to $k = 4$. The x axis represents the size of each partition n , while y axis represents the quotient. Compared to results from Figure 4, we can see that on this dataset, the difference between B_M , C_M , E_M , $E_M(10)$ and $F_M(100)$ are becoming almost negligible when n increases.

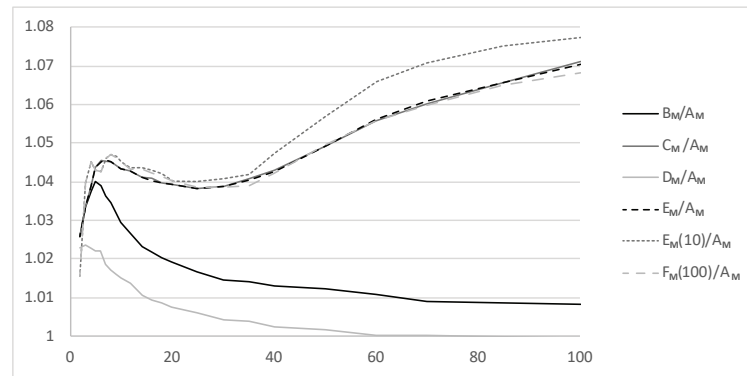


Figure 6. This diagram depicts quotients with A_M for the second subset of the third dataset (corresponding to $k = 3$). Compared to results from Figure 4, we can see that on this subset, the Algorithms C_M , E_M , $E_M(10)$, and $F_M(100)$ give substantially better results than B_M and D_M .

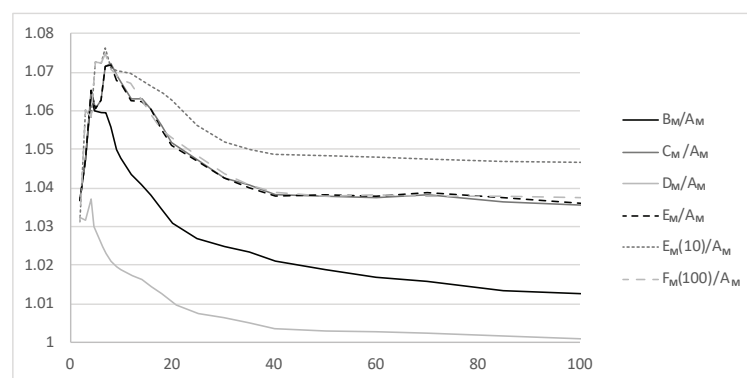


Figure 7. This diagram depicts quotients with A_M on the second subset of the third dataset (corresponding to $k = 4$). Algorithm $E_M(10)$ outperforms the others. However, the ratios seem to stabilise at a constant factor.

Remark 2. When experimentally testing the performance of heuristics, it is well known that random instances are often among the simplest and many heuristics perform remarkably well on such datasets. It is often difficult to generate instances that are hard, or better to say, that are hard for a specific heuristic algorithm. The dataset generated using a nonuniform distribution of weights is obviously harder for the algorithms B and D . It may be of some interest to see what distribution of weights may show further significant differences in performance among the algorithms that are of interest here.

4.4. Third Experiment—Dataset Crama and Spieksma (1992)

In this subsection, we compare Algorithms A_m , B_m , C_m , D_m , E_m , $E_m(10)$, and $F_m(100)$ with the optimal solution of 3-AP on the problems from the second database, which were taken from [30], see also Section 4.1.

For these instances, we know the optimum value of 3-AP (denoted by OPT), so we can report the relative gap, defined by

$$\delta = 100 \cdot \frac{\text{val} - OPT}{OPT}.$$

The results are given in Table 3 for non-binary problems and in Table 4 for binary problems.

Table 3. Comparison of Algorithms A_m , B_m , C_m , D_m , E_m , $E_m(10)$, and $F_m(100)$ on the first group of instances of 3-AP from [15,30]. Column 2 contains the optimum value of the problem, as reported in [30]. For each algorithm, we report the value that it returns. Average relative errors δ and average computation times are given in the last two rows. Algorithms C_m , E_m , $E_m(10)$, and $F_m(100)$ have the best performance and, on average, differ from the optimal solution by 0.1% or less (see the last row).

Problem	OPT	A_m	B_m	C_m	D_m	E_m	$E_m(10)$	$F_m(100)$
3DA198N1	2662	2696	2669	2663	2669	2663	2663	2663
3DA198N2	2449	2498	2467	2458	2467	2458.4	2457.1	2457.4
3DA198N3	2758	2811	2778	2764	2778	2764	2764	2764
3DA99N1	1608	1617	1617	1608	1617	1608.0	1608.0	1608.0
3DA99N2	1401	1420	1411	1402	1415	1402.0	1402.0	1402.0
3DA99N3	1604	1612	1612	1604	1612	1604.0	1604.0	1604.0
3DI198N1	9684	9830	9765	9695	9765	9693.3	9689.2	9689.8
3DI198N1	8944	9132	9121	8949	9177	8949.7	8947.4	8948.4
3DI198N3	9745	9930	9876	9750	9876	9749.6	9747.6	9748.5
3DIJ99N1	4797	4882	4839	4800	4882	4801.3	4798	4799.3
3DIJ99N2	5067	5145	5136	5074	5145	5071.8	5069.6	5071.1
3DIJ99N3	4287	4338	4338	4291	4371	4290.5	4287.8	4289.6
$\bar{\delta}[\%]$	0	1.47	0.92	0.10	1.15	0.10	0.07	0.08
Time	-	1	2.9	11.8	2.3	9.0	89.1	150.9

Table 4. Numerical result for Algorithms A_m , B_m , and C_m on the second group of instances from [30] (called binary graphs). The optimum values OPT are taken from [30], and the values val are computed by Algorithms A_m , B_m , C_m , D_m , E_m , $E_m(10)$, and $F(100)$. Average relative errors δ and average computation times are given in the last two rows. We can see that $F_m(100)$ has the best performance.

Problem	OPT	A_m	B_m	C_m	D_m	E_m	$E(10)$	$F_m(100)$
3D _m 198N1	286	298	294	287	295	286.5	286.0	286.4
3D _m 198N2	286	294	293	286	294	286.2	286.0	286.2
3D _m 198N3	282	294	294	285	294	284.9	284.3	283.7
3D _m 299N1	133	140	134	134	134	134.0	134.0	133.4
3D _m 299N2	131	139	137	134	137	134.0	134.0	133.0
3D _m 299N3	131	136	136	132	136	132.0	132.0	131.0
$\bar{\delta}[\%]$	0	4.41	3.11	0.87	3.22	0.84	0.77	0.45
Time	0	1	2.7	6.8	2.4	5.4	54.0	89.7

For non-binary graphs, with results presented in Table 3, Algorithms C_m , E_m , $E_m(10)$, and $F(100)$ have, as expected, the best performance and on average differ from the optimal solution by 0.1% or less (see last row in Table 3). In addition, they are in some cases also able to find the optimal solution.

For binary graphs, with results presented in Table 4, we can observe that the relative performances are, due to the low weight sum, worse than those of non-binary graphs. As the problems are binary, a solution that differs from the optimal in one element may have, due to small total weight of the assignments, a considerably large relative error. As in the case of non-binary graphs, C_m , E_m , $E_m(10)$, and $F(100)$ outperform A_m , B_m , and D_m . Algorithm $F_m(100)$ finds the optimal solution in most cases (see the last column), and Algorithms C_m , and E_m find the optimal solution in some cases.

We point out that these algorithms are fast. Our implementation, which could be further optimised, takes a fraction of a second (on a 3.0 Ghz PC) on each of these instances. Relative computation times (relative to algorithm A) and average relative errors (compared to known optimal solutions) are evident from Figure 8.

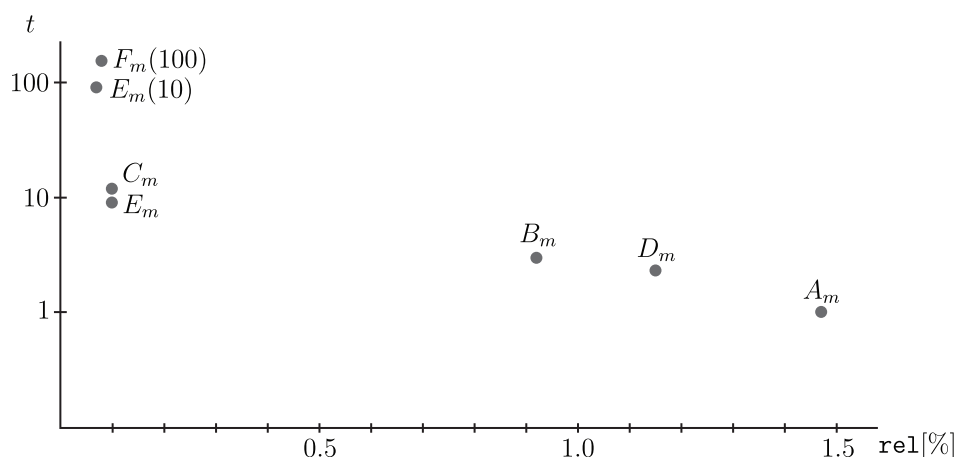


Figure 8. This diagram contains graphical representations of average (normalized) times (in logarithmic scale) needed for non-binary instances from [30] computed in Table 3 and relative errors rel (with respect to the optimal value OPT). Algorithms A_m , B_m , C_m , D_m , and E_m are considered fast, while $E_m(10)$ and $F_m(100)$ are comparably slow.

5. Summary and Conclusions

We have introduced Algorithms A, B, C, D, E, and F to approximately solve (1). The algorithms are all based on extensive use of the Hungarian algorithm and thus arise as natural improvements of Algorithm A from [28]. Algorithms A, B, C, and D are in principle deterministic, whereas Algorithms E and F incorporate randomization. We implemented the algorithms in Python and evaluated them on three benchmark datasets. Numerical tests show that new algorithms in minimization or maximization variant, in terms of solution quality, outperform A on all of the chosen datasets. Summing up, our study shows that multiple usage of the classic Hungarian method can provide very tight solutions for (1), in some cases even an optimal solution.

Another important issue when regarding algorithms' performance is computational time. For smaller instances, E has relatively good speed and on average misses the optimal solution by merely 0.1%, thus, we propose it as our method of choice. Among the deterministic algorithms, our study suggests using Algorithm C. However, we wish to note that when we consider large instances of (1), both in number of partitions and in size of each partition, we must be very careful how often we will actually run the Hungarian method because many repetitions of the Hungarian method substantially increase computation time. The main goal of the reported research was to explore the potential of the Hungarian algorithm for solving the k -assignment problem. We have designed several heuristics based on the Hungarian method that have shown to be competitive. While, on one hand, some of our algorithms provide very good (near optimal or even optimal) results in a short time, we also designed two heuristics based on local search [31–33]. Local search type heuristics improve the quality of solutions over time and may converge to the optimal solution. This type of heuristics are very useful when the quality of solutions is more important than computational time. We believe that further development of a multistart local search heuristics based on the Hungarian algorithm may lead to a very competitive heuristics for (1) with hopefully competitive fast convergence to optimal solutions.

In the future, a more comprehensive experimental study of local search based on the Hungarian algorithm may be a very promising avenue of research.

Author Contributions: Funding acquisition, J.P.; Methodology, J.Ž.; Software, B.G.; Supervision, J.P. and J.Ž.; Writing—original draft, B.G., T.N. and D.R.P.; Writing—review & editing, J.P. and J.Ž. All authors have read and agreed to the published version of the manuscript.

Funding: This research is funded in part by Javna Agencija za Raziskovalno Dejavnost RS, grants: J1-8155, J1-1693, P2-0248, and J2-2512.

Acknowledgments: The authors wish to thank to three anonymous reviewers for a number of constructive comments that helped us to considerably improve the presentation.

Conflicts of Interest: The authors declare have no conflicts of interest.

References

1. Gligorijević, V.; Malod-Dognin, N.; Pržulj, N. Integrative methods for analyzing big data in precision medicine. *Proteomics* **2016**, *16*, 741–758. [\[CrossRef\]](#) [\[PubMed\]](#)
2. Gligorijević, V.; Malod-Dognin, N.; Pržulj, N. Fuse: Multiple network alignment via data fusion. *Bioinformatics* **2015**, *32*, 1195–1203. [\[CrossRef\]](#) [\[PubMed\]](#)
3. Malod-Dognin, N.; Petschnigg, J.; Windels, S.F.L.; Povh, J.; Hemmingway, H.; Ketteler, R.; Pržulj, N. Towards a data-integrated cell. *Nat. Commun.* **2019**, *10*, 805. [\[CrossRef\]](#) [\[PubMed\]](#)
4. Balas, E.; Saltzman, M.J. An algorithm for the three-index assignment problem. *Oper. Res.* **1991**, *39*, 150–161. [\[CrossRef\]](#)
5. Burkard, R.; Dell’Amico, M.; Martello, S. *Assignment Problems: Revised Reprint*; SIAM-Society of Industrial and Applied Mathematics: Philadelphia, PA, USA, 2012; Volume 106. [\[CrossRef\]](#)
6. Burkard, R.E.; Rudolf, R.; Woeginger, G.J. Three-dimensional axial assignment problems with decomposable cost coefficients. *Discret. Appl. Math.* **1996**, *65*, 123–139. [\[CrossRef\]](#)
7. Frieze, A.M. Complexity of a 3-dimensional assignment problem. *Eur. J. Oper. Res.* **1983**, *13*, 161–164. [\[CrossRef\]](#)
8. Spieksma, F. Multi Index Assignment Problems: Complexity, Approximation, Applications. In *Nonlinear Assignment Problems*; Springer: Boston, MA, USA, 2000; pp. 1–12. [\[CrossRef\]](#)
9. Kuroki, Y.; Matsui, T. An approximation algorithm for multidimensional assignment problems minimizing the sum of squared errors. *Discret. Appl. Math.* **2009**, *157*, 2124–2135. [\[CrossRef\]](#)
10. Grundel, D.A.; Krokhmal, P.A.; Oliveira, C.A.S.; Pardalos, P.M. On the number of local minima for the multidimensional assignment problem. *J. Comb. Optim.* **2007**, *13*, 1–18. [\[CrossRef\]](#)
11. Kuhn, H.W. The Hungarian Method for the Assignment Problem. *Nav. Res. Logist. Q.* **1955**, *2*, 83–97. [\[CrossRef\]](#)
12. Karp, R.M. Reducibility Among Combinatorial Problems. In *Complexity of Computer Computations*; Plenum: New York, NY, USA, 1972; pp. 85–103. [\[CrossRef\]](#)
13. Hurkens, C.A.J.; Schrijver, A. On the size of systems of sets every t of which have an SDR, with an application to the worst-case ratio of heuristics for packing problems. *SIAM J. Discret. Math.* **1989**, *2*, 68–72. [\[CrossRef\]](#)
14. Arkin, E.; Hassin, R. On local search for weighted packing problems. *Math. Oper. Res.* **1998**, *23*, 640–648. [\[CrossRef\]](#)
15. Crama, Y.; Spieksma, F. Approximation algorithms for three-dimensional assignment problems with triangle inequalities. *Eur. J. Oper. Res.* **1992**, *60*, 273–279. [\[CrossRef\]](#)
16. Aiex, R.M.; Resende, M.G.C.; Pardalos, P.M.; Toraldo, G. Grasp with path relinking for three-index assignment. *Inform. J. Comput.* **2005**, *17*, 224–247. [\[CrossRef\]](#)
17. Huang, G.; Lim, A. A hybrid genetic algorithm for the Three-Index Assignment Problem. *Eur. J. Oper. Res.* **2006**, *172*, 249–257. [\[CrossRef\]](#)
18. Gutin, G.; Karapetyan, D. Local Search Heuristics for the Multidimensional Assignment Problem. *J. Heuristics* **2011**, *17*, 201–249. [\[CrossRef\]](#)
19. Karapetyan, D.; Gutin, G.; Goldengorin, B. Empirical evaluation of construction heuristics for the multidimensional assignment problem. *arXiv* **2009**, arXiv:0906.2960.
20. Jiang, H.; Zhang, S.; Ren, Z.; Lai, X.; Piao, Y. Approximate Muscle Guided Beam Search for Three-Index Assignment Problem. *Adv. Swarm Intell. Lect. Notes Comput. Sci.* **2014**, *8794*, 44–52. [\[CrossRef\]](#)
21. Karapetyan, D.; Gutin, G. A New Approach to Population Sizing for Memetic Algorithms: A Case Study for the Multidimensional Assignment Problem. *Evol. Comput.* **2011**, *19*, 345–371. [\[CrossRef\]](#)
22. Valencia, C.E.; Zaragoza Martinez, F.J.; Perez, S.L.P. A simple but effective memetic algorithm for the multidimensional assignment problem. In *Proceedings of the 14th International Conference on Electrical Engineering, Computing Science and Automatic Control (CCE)*, Mexico City, Mexico, 20–22 October 2017; pp. 1–6. [\[CrossRef\]](#)

23. Li, J.; Tharmarasa, R.; Brown, D.; Kirubarajan, T.; Pattipati, K.R. A novel convex dual approach to three-dimensional assignment problem: Theoretical analysis. *Comput. Optim. Appl.* **2019**, *74*, 481–516. [[CrossRef](#)]
24. O’Leary, B. Don’t be Greedy, be Neighborly, a new assignment algorithm. In Proceedings of the 2019 IEEE Aerospace Conference, Big Sky, MT, USA, 2–9 March 2019; pp. 1–8.
25. Medvedev, S.N.; Medvedeva, O.A. An Adaptive Algorithm for Solving the Axial Three-Index Assignment Problem. *Autom. Remote Control* **2019**, *80*, 718–732. [[CrossRef](#)]
26. Pentico, D. Assignment problems: A golden anniversary survey. *Eur. J. Oper. Res.* **2007**, *176*, 774–793. [[CrossRef](#)]
27. Walteros, J.; Vogiatzis, C.; Pasiliao, E.; Pardalos, P. Integer programming models for the multidimensional assignment problem with star costs. *Eur. J. Oper. Res.* **2014**, *235*, 553–568. [[CrossRef](#)]
28. He, G.; Liu, J.; Zhao, C. Approximation algorithms for some graph partitioning problems. In *Graph Algorithms and Applications 2*; World Scientific: Singapore, 2004; pp. 21–31.
29. Mladenović, N.; Hansen, P. Variable neighborhood search. *Comput. Oper. Res.* **1997**, *24*, 1097–1100. [[CrossRef](#)]
30. Spieksma, F.C.R. Instances of the 3-Dimensional Assignment Problem. Available online: <https://www.win.tue.nl/~fspieksma/instancesEJOR.htm> (accessed on 15 February 2019).
31. Aarts, E.H.L.; Lenstra, J.K. (Eds.) *Local Search in Combinatorial Optimization*; Wiley-Interscience Series in Discrete Mathematics and Optimization; Wiley-Interscience: Hoboken, NJ, USA, 1997.
32. Talbi, E. *Metaheuristics: From Design to Implementation*; John Wiley & Sons: Hoboken, NJ, USA, 2009.
33. Žerovnik, J. Heuristics for NP-hard optimization problems—Simpler is better !? *Logist. Sustain. Transp.* **2015**, *6*, 1–10. [[CrossRef](#)]

Publisher’s Note: MDPI stays neutral with regard to jurisdictional claims in published maps and institutional affiliations.



© 2020 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<http://creativecommons.org/licenses/by/4.0/>).