On Some Nonlinear Assignment Problems

by

Vladyslav Sokol

M.Sc., Kyiv Polytechnic Institute, 2012
B.Sc., Kyiv Polytechnic Institute, 2010

Thesis Submitted in Partial Fulfillment of the
Requirements for the Degree of
Doctor of Philosophy

in the
School of Computing Science
Faculty of Applied Sciences

SIMON FRASER UNIVERSITY
Spring 2018

Approval

Name:	**Vladyslav Sokol**
Degree:	**Doctor of Philosophy (Computer Science)**
Title:	***On Some Nonlinear Assignment Problems***
Examining Committee:	**Chair:** Faraz Hach Research Associate

Binay Bhattacharya
Co-Supervisor
Professor

Abraham Punnen
Co-Supervisor
Professor
Department of Mathematics

Ramesh Krishnamurti
Supervisor
Professor

Oliver Schulte
Internal Examiner
Professor

Zhipeng Lu
External Examiner
Full Professor, Director
School of Computer Science
Huazhong University of Science and Technology

Date Defended: 27 February 2018

Abstract

Linear assignment problem (commonly referred to as just *assignment problem*) is a fundamental problem in combinatorial optimization. The goal is to assign n workers to do n jobs so that the linear sum of corresponding costs is minimized. The linear assignment problem is thoroughly studied and has a $O(n^3)$ solution with Hungarian algorithm. Nevertheless, a wide range of applications involving assignments are naturally modeled with more complex objective functions (for example quadratic sum as in *quadratic assignment problem*), and are much more computationally challenging.

In this thesis we discuss our results on the *bilinear assignment problem*, which generalizes the quadratic assignment problem, and is also motivated by several unique applications. The focus is on computational complexity, solvable special cases, approximations, linearizations as well as local search algorithms and other heuristic approaches for the problem. We also present our results on few applied projects, where modelling the underlying problem as a nonlinear assignment was instrumental.

Keywords: assignment problem; quadratic assignment problem; computational complexity; polynomially solvable cases; approximation; PTAS; domination analysis; exponential neighborhood; heuristic; variable neighborhood search; integer programming; linearization; ridesharing; vehicle routing

Acknowledgements

I thank Ramesh Krishnamurti for the introduction to integer programming techniques and research process at SFU Computer Science.

I thank Binay Bhattacharya for the breadth of my theoretical and practical knowledge and the provided opportunity to advance my studies with a doctorate.

I thank Abraham Punnen for the given ability to study interesting problems to the extent and detail that i have never had before.

And lastly, thank you to Ante Custic for help with many proofs and for countless discussions about research and life.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Assignments and linear assignment problem

Assignment problems are concerned with assigning n jobs (items, facilities) to n workers (items, locations). Given a $n \times n$ matrix of costs c_{ij} and n^2 binary decision variables, where $x_{ij} = 1$ stands for assigning job i to worker j, we can write the following:

$$\text{Minimize} \quad \sum_{i=1}^{n} \sum_{j=1}^{n} c_{ij} x_{ij} \tag{1.1}$$

$$\text{subject to} \quad \sum_{j=1}^{n} x_{ij} = 1 \qquad i = 1, 2, \ldots, n, \tag{1.2}$$

$$\sum_{i=1}^{n} x_{ij} = 1 \qquad j = 1, 2, \ldots, n, \tag{1.3}$$

$$x_{ij} \in \{0, 1\} \qquad i, j = 1, 2, \ldots, n. \tag{1.4}$$

Problem (1.1)-(1.4) is called *linear assignment problem (LAP)*, as (1.1) is a linear sum w.r.t. variables. (1.2)-(1.3) are called *assignment constraints* and correspond to permutations of n-element set. Let Π be the set of all permutations on $\{1, 2, \ldots, n\}$. Then LAP can be also written in permutation form:

$$\min_{\pi \in \Pi} \sum_{i=1}^{n} c_{i\,\pi(i)}. \tag{1.5}$$

If, say, (1.3) is removed from the formulation, we will be dealing with the case where both $x_{i_1 j} = 1$ and $x_{i_2 j} = 1$ for $i_1 \neq i_2$ can be present in a feasible solution. This type of constraints are *semi-assignment constraints*, and together with (1.1) define *linear semi-*

assignment problem:

$$\text{Minimize} \quad \sum_{i=1}^{n}\sum_{j=1}^{m} c_{ij}x_{ij} \tag{1.6}$$

$$\text{subject to} \quad \sum_{j=1}^{m} x_{ij} = 1 \qquad i = 1,2,\ldots,n, \tag{1.7}$$

$$x_{ij} \in \{0,1\} \qquad i = 1,2,\ldots,n,\ j = 1,2,\ldots,m. \tag{1.8}$$

This problem admits a trivial $O(nm)$ solution, but the terminology will be useful for later parts of the survey.

LAP is an important problem in combinatorial optimization with numerous applications dealing with assignments of discrete objects. The famous Hungarian algorithm provides an optimal solution to LAP in $O(n^3)$ running time [90]. Many applications involving assignments require more complicated interactions between variables to correctly capture all processes in a model. The best example is Quadratic Assignment Problem, which gained the most attention among nonlinear assignment problems from the practitioners and Operations Research community. Generally speaking, this additional complexity in the objective function or/and assignments structures brings the problem into the NP-hard category, and thus will force an extra effort to find an optimal (or even near optimal) solutions. However, occasionally, a special information about an input of the problem will allow for a more efficient solution approach. In this chapter we will try to organize our knowledge about nonlinear assignment problems and establish connections between them in terms of generalization/special case relations. We will occasionally refer to several other seminal surveys in the area, as we do not hope for a perfectly detailed review of all the related research that has been done. However, for several of the nonlinear assignment problems new results have been recently obtained, and they will be discussed in such survey format for the first time.

An outline of this chapter is like this. Each section will talk about an important problem or set of problems and will follow the same structure. First a formulation will be presented to demonstrate the difference with LAP and connections to other assignment problems. Next we will give a quick overview of applications that sprung interest for the problem. Following, will be a discussion of results on complexity of the problem together with its special cases, approximability results, average and worst case analysis of solutions. Finally, we will mention algorithmic approaches that researchers attempted for those problems.

1.2 Overview of nonlinear assignment problems

1.2.1 Quadratic assignment and related problems

Quadratic assignment problem (QAP), introduced in [87], was dealing with assigning n facilities to n locations. The cost function here, besides the linear costs, also contains a part that depends on pairs of variables and is proportional to the value of the flow between two facilities, multiplied by the distance between two locations. Using same decision variables that we used for describing LAP, this *Koopmans-Beckmann* form of QAP can be stated like this:

$$\begin{aligned}
\text{minimize} \quad & \sum_{i=1}^{n}\sum_{j=1}^{n}\sum_{k=1}^{n}\sum_{l=1}^{n} f_{ik}d_{jl}x_{ij}x_{kl} + \sum_{i=1}^{n}\sum_{j=1}^{n} c_{ij}x_{ij}, && (1.9)\\
\text{subject to} \quad & \sum_{j=1}^{n} x_{ij} = 1 \qquad i = 1, 2, \dots, n, && (1.10)\\
& \sum_{i=1}^{n} x_{ij} = 1 \qquad j = 1, 2, \dots, n, && (1.11)\\
& x_{ij} \in \{0, 1\} \qquad i, j = 1, 2, \dots, n, && (1.12)
\end{aligned}$$

where $F = f_{ik}$ and $D = d_{jl}$ are $n \times n$ flow and distance matrices respectively. Objective function depends quadratically on decision variables x_{ij}, hence the *quadratic* in the name of the problem. Notice that QAP constraints (1.10) - (1.12) are identical to constraints of the linear assignment problem discussed above.

QAP in more general *Lawler's* [94] form is to

$$\begin{aligned}
\text{minimize} \quad & \sum_{i=1}^{n}\sum_{j=1}^{n}\sum_{k=1}^{n}\sum_{l=1}^{n} q_{ijkl}x_{ij}x_{kl}, && (1.13)\\
\text{subject to} \quad & \sum_{j=1}^{n} x_{ij} = 1 \qquad i = 1, 2, \dots, n, && (1.14)\\
& \sum_{i=1}^{n} x_{ij} = 1 \qquad j = 1, 2, \dots, n, && (1.15)\\
& x_{ij} \in \{0, 1\} \qquad i, j = 1, 2, \dots, n. && (1.16)
\end{aligned}$$

Instead of flow and distance matrices, we are using an array Q of size $n \times n \times m \times m$. Koopmans-Beckmann QAP can be transformed into Lawler's form by setting:

$$\begin{aligned}
& q_{ijkl} = f_{ik}d_{jl} \qquad i, j, k, l = 1, 2, \dots, n;\ i \neq k,\ j \neq l && (1.17)\\
& q_{ijij} = f_{ii}d_{jj} + c_{ij} \qquad i, j = 1, 2, \dots, n. && (1.18)
\end{aligned}$$

The permutation formulation of QAP is

$$\min_{\pi\in\Pi}\sum_{i=1}^{n}\sum_{k=1}^{n} q_{i\,\pi(i)\,k\,\pi(k)}, \tag{1.19}$$

where Π is the set of all permutations on $\{1, 2, \ldots, n\}$.

Note that for QAP we can assume without loss of generality that $q_{ijil} = 0$ when $j \neq l$ and similarly $q_{ijkj} = 0$ when $i \neq k$. Also, by reassigning all $q_{ijkl} = q_{klij} = \frac{q_{ijkl}+q_{klij}}{2}$ we can make n^2 by n^2 matrix Q symmetric, while maintaining equivalence to the original problem.

Numerous real-world applications established QAP as the focus of study for many from the Operations Research, Combinatorial Optimization and related communities. A non-exhaustive list of problems that could be modeled as QAP include: campus building planning, minimizing the number of connections in a backboard wiring problem, hospital and forest planning, computer manufacturing, scheduling, process communications, turbine balancing, typewriter keyboard design, turbine runner problem, ranking of archeological data, ranking of a team in a relay race, scheduling of parallel production lines, analysis of chemical reactions for organic compounds, arrangement of numbers around a dartboard under risk maximization, the arrangement of probes on microarray chips, combinatorial data analysis and shunting of trams in a storage yard. A detailed description and references to applications can be viewed in a great variety of surveys dedicated to and including QAP [25, 19, 95, 113, 21, 103].

QAP is also a generalization of several other important problems, such as: traveling salesman problem, minimum weight feedback arc set problem, graph partitioning and maximum clique problem, graph isomorphism and graph packing problems. Hence, it is possible to model their applications as QAP too.

Complexity and solvable cases

QAP is NP-hard, as TSP is its special case. It is not possible to find a constant factor approximation for the QAP in polynomial time, unless P = NP [116]. This results holds even for Koopmans-Beckmann QAP with F, D, C satisfying triangle inequality [110]. However, maximization version of the QAP in which matrix D satisfies the triangle inequality admits a 1/4-approximation [9]. A special case of Koopmans-Beckmann QAP called linear dense arrangement problem, where matrix F is the distance matrix of n points which are regularly spaced on a line and the number of 1-entries in D is at least kn^2 for some constant k, is shown to have polynomial-time approximation scheme [10].

If we are to minimize the maximum cost instead of the sum (1.13), we would be dealing with *quadratic bottleneck assignment problem* [124].

It is known [19] that the average value of all solutions to a given QAP instance can be computed as:

$$\mathcal{A} = \frac{1}{n(n-1)} \sum_{i,j=1}^{n} \sum_{k \neq i} \sum_{l \neq j} q_{i\,j\,k\,l} + \frac{1}{n} \sum_{i,j=1}^{n} q_{i\,j\,i\,j}. \tag{1.20}$$

With respect to domination analysis, authors of [67] presented a heuristic with no worse than the average value performance guarantee, that has the domination number at least $\frac{n!}{\beta^n}$, for any $\beta > 1$. For every prime power n, the domination number of this heuristic is shown to be at least $(n-2)!$.

The time complexity of 2- and 3-exchange methods for the QAP is exponential in the worst case [100, 103]. Deciding whether a given local optimum solution of the QAP is also global optimum is NP-complete [101]. It is shown that the optimal value for the 2-exchange local search will be at most n times larger than the average [6]. The solution with this value guarantee will be found by 2-exchange local search in polynomial number of steps.

Solution approaches

QAP have been attempted to be solved by majority of the popular exact and heuristic methods. In terms of branch-and-bound [102, 68, 7, 1, 30] researchers considered different branching (pair assignment, relative positioning, polytomic branching), bounding and exploration strategies. A large variety of valid inequalities inducing facets was proposed for branch-and-cut [14]. See [19] for more details on exact and heuristic solution methods and an extensive survey on metaheuristics [43] applied to QAP.

Several linearizations were proposed to reformulate quadratic assignment problem as an integer linear program [79, 48, 94, 1]. Following this route, one can attempt to optimally solve this (larger in terms of number of variables and/or number of constraints) integer programming linearization, or relax the integrality constraints and use the relaxation solution as a bound for other methods.

Quadratic semi-assignment problem

Similarly to semi-assignment case of its linear counterpart, *quadratic semi-assignment problem* [60] does not have constraint (1.15) and is dealing with assigning potentially larger number n of facilities to m locations, $n \geq m$, such that the quadratic cost is minimized:

$$\begin{aligned}
\text{minimize} \quad & \sum_{i=1}^{n} \sum_{j=1}^{m} \sum_{k=1}^{n} \sum_{l=1}^{m} q_{ijkl} x_{ij} x_{kl}, && && (1.21)\\
\text{subject to} \quad & \sum_{j=1}^{m} x_{ij} = 1 && i = 1, 2, \ldots, n, && (1.22)\\
& x_{ij} \in \{0, 1\} && i = 1, 2, \ldots, n,\ j = 1, 2, \ldots, m. && (1.23)
\end{aligned}$$

The quadratic semi-assignment problem is NP-hard, as it generalizes QAP with a simple transformation. Namely, by adding a large number to all elements q_{ijil} where $j \neq l$ of Q, one can prohibit two facilities to be assigned to the same location. In this way an algorithm that solves quadratic semi-assignment problem will, therefore obtain an optimal solution to the original QAP.

This problem has been considered as a model for various applications such as supply support for space bases, schedule synchronization in transit networks, task scheduling on distributed computing systems and minimizing the mean flow time in parallel processors scheduling. Reader is referred to survey [19] for references of applications as well as lower bounds, polyhedral studies and admissible transformations for this problem.

In [15] author considers a class of polynomially solvable special cases of the quadratic semi-assignment problem. These cases deal with an input array Q representing a *flow graph* ($\sum_{j=1}^{m} \sum_{l=1}^{m} q_{ijkl} = 1$ when i and k are connected with an edge and 0 otherwise). Authors provide an $O(nm^2)$ algorithm to solve such assignment problem when Q represents flow graph of a tree. If Q represents a flow graph that can be reduced to a single node by means of tail, series and parallel reductions, then a $O(nm^3)$ algorithm to find an optimal solution is given [28, 96].

In [13] authors propose to solve the quadratic semi-assignment problem using lower-first branch-and-bound algorithm that relies on Lagrangean relaxation bounds. Computational results are reported for instances of sizes up to $n = 101$, $m = 10$. In terms of heuristics researchers described adaptations of simulated annealing and tabu search [41, 132, 42].

Cubic and quartic assignment problems

In the beginning of the chapter we considered extending the objective function from linear in LAP to quadratic in QAP. To generalize we can define *m-tic assignment problem* [94] as minimizing:

$$\sum_{i_1=1}^{n} \sum_{i_2=1}^{n} \cdots \sum_{i_{2m}=1}^{n} q_{i_1 i_2 \ldots i_{2m}} x_{i_1 i_2} x_{i_3 i_4} \ldots x_{i_{2m-1} i_{2m}}, \tag{1.24}$$

subject to assignment constraints as in (1.2)-(1.4).

To this point only 3- and 4-tic problems (called cubic and quartic or bi-quadratic respectively) have found applications, according to the literature. Cubic assignment problem appears as a model for finding the minimum shunting of trams in a storage yard, whether, the quartic assignment problem is useful in VLSI synthesis [26]. Authors of the paper provide several results on lower bounds and probabilistic behavior of the problem. Both problems have found use in improving existing lower bounds for QAP [1].

Reported solution methods for quartic assignment problem, due to complexity, are limited to heuristics such as combinations of 2-exchange local searches, simulated annealing and tabu search [20].

1.2.2 Problems with multiple assignments

Bilinear Assignment Problem

Let $Q = (q_{ijk\ell})$ be an $m \times m \times n \times n$ array, $C = (c_{ij})$ be an $m \times m$ matrix, and $D = (d_{k\ell})$ be an $n \times n$ matrix. Then the *bilinear assignment problem* (BAP) is to

$$\text{Minimize} \quad \sum_{i=1}^{m}\sum_{j=1}^{m}\sum_{k=1}^{n}\sum_{\ell=1}^{n} q_{ijk\ell}x_{ij}y_{k\ell} + \sum_{i=1}^{m}\sum_{j=1}^{m} c_{ij}x_{ij} + \sum_{k=1}^{n}\sum_{\ell=1}^{n} d_{k\ell}y_{k\ell} \tag{1.25}$$

$$\text{subject to} \quad \sum_{j=1}^{m} x_{ij} = 1 \qquad i = 1, 2, \ldots, m, \tag{1.26}$$

$$\sum_{i=1}^{m} x_{ij} = 1 \qquad j = 1, 2, \ldots, m, \tag{1.27}$$

$$\sum_{\ell=1}^{n} y_{k\ell} = 1 \qquad k = 1, 2, \ldots, n, \tag{1.28}$$

$$\sum_{k=1}^{n} y_{k\ell} = 1 \qquad \ell = 1, 2, \ldots, n, \tag{1.29}$$

$$x_{ij},\ y_{k\ell} \in \{0, 1\} \qquad i, j = 1, 2, \ldots, m, \quad k, \ell = 1, \ldots, n. \tag{1.30}$$

Let $\mathcal{X}$ be the set of all $m \times m$ 0-1 matrices satisfying (1.26) and (1.27) and $\mathcal{Y}$ be the set of all $n \times n$ 0-1 matrices satisfying (1.28) and (1.29). Also, let $\mathcal{F}$ be the set of all feasible solutions of BAP. Note that $|\mathcal{F}| = m!n!$. An instance of the BAP is fully defined by the 3-tuple of cost arrays (Q, C, D). Let $M = \{1, 2, \ldots, m\}$ and $N = \{1, 2, \ldots, n\}$. Without loss of generality we assume that $m \leq n$. The objective function of BAP is denoted by $f(\mathbf{x}, \mathbf{y})$ where $\mathbf{x} = (x_{ij}) \in \mathcal{X}$ and $\mathbf{y} = (y_{ij}) \in \mathcal{Y}$. The quadratic part of the objective function, i.e. $\sum_{i=1}^{m}\sum_{j=1}^{m}\sum_{k=1}^{n}\sum_{\ell=1}^{n} q_{ijk\ell}x_{ij}y_{k\ell}$, is denoted by $\bar{f}(\mathbf{x}, \mathbf{y})$. It may be noted that in BAP, constraints (1.30) can be replaced by $0 \leq x_{ij} \leq 1$ and $0 \leq y_{k\ell} \leq 1$ for $i, j = 1, \ldots, m$, and $k, \ell = 1, \ldots, n$.

Being a generalization of the well studied problems QAP and 3AP, applications of these models translate into applications of BAP. Further, Zikan [134] used a model equivalent to BAP to solve track initialization in the multiple-object tracking problem, Tsui and Chang [128, 129] used BAP to model a dock door assignment problem, and Torki, Yajima and Enkawa [126] used BAP to obtain heuristic solutions to QAP with a small rank Q.

Bipartite Quadratic Assignment Problem

Closely related to quadratic semi-assignment problem, *bipartite quadratic assignment problem (BQAP)* can be written as the following integer program:

$$\text{Minimize} \quad \sum_{i=1}^{m}\sum_{j=1}^{m}\sum_{k=1}^{n}\sum_{l=1}^{n} q_{ijkl}x_{ij}y_{kl} + \sum_{i=1}^{m}\sum_{j=1}^{m} c_{ij}x_{ij} + \sum_{k=1}^{n}\sum_{l=1}^{n} d_{kl}y_{kl} \tag{1.31}$$

$$\text{subject to} \quad \sum_{i=1}^{m} x_{ij} = 1 \qquad j = 1, 2, \ldots, m, \tag{1.32}$$

$$\sum_{l=1}^{n} y_{kl} = 1 \qquad k = 1, 2, \ldots, n, \tag{1.33}$$

$$x_{ij},\ y_{kl} \in \{0, 1\} \qquad i, j = 1, \ldots, m,\ k, l = 1, \ldots, n. \tag{1.34}$$

Here we have real valued four dimensional array $Q = (q_{ijkl})$ of size $m \times m \times n \times n$, an $m \times m$ matrix $C = (c_{ij})$, and an $n \times n$ matrix $D = (d_{kl})$.

Applications for BQAP predominantly lie in the area of facility location. Moreover, BQAP is a proper generalization of QAP [109] and can be used as a model for variety of corresponding applications.

It is shown that BQAP can be considered as a special case of quadratic semi-assignment problem [109]. However, authors use a more general quadratic programming model to explore complexity. For this problem constraints are parametrized and may not be of an assignment kind. Following, NP-hardness is shown for several special cases.

In [37] authors study BQAP from the point of view of complexity, domination analysis and solvable special cases. The closed formula to calculate the average value $\mathcal{A}$ of all solutions is presented as:

$$\mathcal{A}(Q, c, d) = \frac{1}{mn}\sum_{i=1}^{m}\sum_{j=1}^{m}\sum_{k=1}^{n}\sum_{\ell=1}^{n} q_{ijk\ell} + \frac{1}{m}\sum_{i=1}^{m}\sum_{j=1}^{m} c_{ij} + \frac{1}{n}\sum_{i=1}^{n}\sum_{j=1}^{n} d_{ij}. \tag{1.35}$$

Every feasible solution with objective function value at most $\mathcal{A}$ has the domination ratio at least $\frac{1}{mn}$. Finding the median objective function value, however, is shown to be NP-hard. Computing a solution whose objective function value is no worse than that of $m^m n^n - \lceil \frac{m}{\alpha} \rceil^{\lceil \frac{m}{\alpha} \rceil} \lceil \frac{n}{\alpha} \rceil^{\lceil \frac{n}{\alpha} \rceil}$ solutions of BQAP, is NP-hard for any rational number $\alpha > 1$. However, a solution with the domination number $\Omega(m^{m-1}n^{n-1} + m^2 n^n + m^m n^2)$ for BQAP, can be found in $O(m^3 n^3)$ time.

Although tools for bilinear programming can be used to solve BQAP, authors [109] look for a better tailored approaches that exploit additional problem structure. Several neighborhood search structures introduced that are based on chains of swaps in the solution as well as solving linear assignment sub-problems. Finally, a Tabu Search algorithm is presented to enhance the performance of local search over described neighborhoods. In [37] it is shown that some heuristics that work well in practice could produce solutions with objective function value worse than the average value of solutions. However, a simple polynomial algorithm that guarantee a solution with objective function value no worse than the average value of solutions is provided in the paper.

1.2.3 Multi-dimensional assignment problems

Axial and planar 3-dimensional assignment problems

Consider the $n \times n \times n$ cost array $A = (a_{ijk})$. The *axial 3-dimensional assignment problem* (3AP) [104] can be written as the following integer linear program:

$$\text{minimize} \quad \sum_{i=1}^{n}\sum_{j=1}^{n}\sum_{k=1}^{n} a_{ijk}x_{ijk} \tag{1.36}$$

$$\text{subject to} \quad \sum_{j=1}^{n}\sum_{k=1}^{n} x_{ijk} = 1 \qquad (i = 1, 2, \ldots, n), \tag{1.37}$$

$$\sum_{i=1}^{n}\sum_{k=1}^{n} x_{ijk} = 1 \qquad (j = 1, 2, \ldots, n), \tag{1.38}$$

$$\sum_{i=1}^{n}\sum_{j=1}^{n} x_{ijk} = 1 \qquad (k = 1, 2, \ldots, n), \tag{1.39}$$

$$x_{ijk} \in \{0, 1\} \qquad (i, j, k = 1, \ldots, n). \tag{1.40}$$

Alternatively, the permutation formulation for 3AP is to find

$$\min_{\pi,\phi\in\Pi} \sum_{i=1}^{n} a_{i\,\pi(i)\,\phi(i)}. \tag{1.41}$$

3AP can be viewed as a special case of bilinear integer programming as discussed in [46], and, in particular, as a special case of BAP. The problem is then to find $\mathbf{x}, \mathbf{y} \in \mathcal{Y}$ such that

$$\sum_{i=1}^{n}\sum_{j=1}^{n}\sum_{k=1}^{n} a_{ijk}x_{ij}y_{jk}$$

is minimized. Now it is easy to see that every 3AP instance with $A = (a_{ijk})$ can be reduced to a BAP instance (Q, O^n, O^n), where $q_{ijk\ell}$ is equal to $a_{ij\ell}$ if $j = k$, and to 0 otherwise.

Axial 3-dimensional assignment problem has applications in areas such as: investment of capital into physical locations over time horizon; ingot temperature stabilization via rolling mill; assembly of printed circuit boards; dynamic facility location; satellite launching; perishable production planning. The bottleneck version of the problem (minimizing maximum cost) has been considered as a model for some time-cost trade-off problems. For more applications and references see [19], [122].

3AP is NP-hard [78] and no polynomial time algorithm can achieve a constant approximation ratio, unless P=NP [33]. This statement holds even for two special cases of 3AP where the cost array A satisfies

$$a_{ijk} = d_{ij} + d_{ik} + d_{jk} \qquad (i, j, k = 1, \ldots, n) \tag{1.42}$$

or

$$a_{ijk} = \min\{d_{ij} + d_{ik}, d_{ij} + d_{jk}, d_{ik} + d_{jk}\} \qquad (i, j, k = 1, \ldots, n) \tag{1.43}$$

for some $n \times n$ array D. However, if every d_{ij}, d_{ik}, d_{jk} satisfy triangle inequality, these special cases of 3AP admit 3/2 and 4/3 approximations respectively [33]. Authors of [123] show that if d_{ij}, d_{ik}, d_{jk} correspond to euclidean distances between $3n$ points in the plane, then 3AP is NP-hard for both minimizing the perimeter (same as 1.42) as well as minimizing the area of the corresponding triangles. The maximization version of 3AP with cost array satisfying (1.42) is polynomially solvable when d_{ij}, d_{ik}, d_{jk} represent polyhedral norm distances between 3n points in $\mathbb{R}^q$ [35]. The general maximization version of 3AP, interestingly, allows for a 1/3 approximation based on the results from [72]. As discovered in [22] 3AP becomes polynomially solvable in case A is a *Monge array*, where for each fixed i (and similarly for each j and k):

$$a_{iuv} + a_{irs} \leq a_{ius} + a_{irl} \qquad \text{for } 1 \leq u < r \leq n,\ 1 \leq v < s \leq n. \tag{1.44}$$

This result is also true for the axial 3-dimensional bottleneck assignment problem if the Monge property of the array is replaced with the bottleneck Monge property [22] or *wedge condition* [81] (see references for details). Another special case of 3AP considered in [23] deals with decomposable costs:

$$a_{ijk} = b_i c_j d_k \qquad (i, j, k = 1, \ldots, n). \tag{1.45}$$

In this form the minimization problem remains NP-hard and hard to approximate, whereas, the maximization problem becomes polynomially solvable. Authors [23] also describe few polynomially solvable special cases of decomposable 3AP based on very restricted structures of arrays B, C, D.

Several exact and heuristic approaches were proposed for the axial 3-dimensional assignment problem. The former include some variations of the primal-dual method, subgradient procedures for Lagrangean relaxation of the problem and branch-and-bound with different branching rules. The later are local search and few popular metaheuristics such as greedy randomized adaptive search procedure with path relinking and hybrid genetic algorithm.

For more detailed review of the polyhedral studies, asymptotic behavior, lower bounds, exact and heuristic algorithms for 3AP the reader is referred to surveys in [19], [122].

The *planar 3-dimensional assignment problem* (planar 3AP) can be written as the following integer linear program:

$$\begin{aligned}
\text{minimize} \quad & \sum_{i=1}^{n}\sum_{j=1}^{n}\sum_{k=1}^{n} a_{ijk}x_{ijk} && && (1.46)\\
\text{subject to} \quad & \sum_{k=1}^{n} x_{ijk} = 1 && (i, j = 1, 2, \ldots, n), && (1.47)\\
& \sum_{i=1}^{n} x_{ijk} = 1 && (j, k = 1, 2, \ldots, n), && (1.48)\\
& \sum_{j=1}^{n} x_{ijk} = 1 && (i, k = 1, 2, \ldots, n), && (1.49)\\
& x_{ijk} \in \{0, 1\} && (i, j, k = 1, \ldots, n). && (1.50)
\end{aligned}$$

The permutation formulation for planar 3AP is to find n mutually distinct permutations $\pi_1, \pi_2, \ldots, \pi_n$ to minimize

$$\sum_{i=1}^{n}\sum_{k=1}^{n} a_{i\,\pi_k(i)\,k}. \tag{1.51}$$

Planar 3AP is NP-hard [47], and related to Latin squares. Applications for planar 3AP include several timetabling and time slot assignment problems.

If the cost array A satisfies $a_{ijk} = a_{ij}$ for $k = 1, \ldots, n$ then the planar 3AP is solvable, with any set of mutually distinct permutations as the solution. Another variation considered involves minimization of slightly different objective function:

$$\sum_{k=1}^{n} \max_{1 \le i \le n} a_{i\,\pi_k(i)\,k}. \tag{1.52}$$

This version of planar 3AP remains NP-hard [114].

Researchers tackled the planar 3AP with several approaches, namely branch-and-bound and tabu search. PTAS for planar 3AP is presented in [51]. For more detailed review of the polyhedral studies, lower bounds and algorithms for planar 3AP and variations see [19].

Two to one assignment problem

Another assignment problem that is somewhat more general than 3AP is *two-to-one assignment problem* (2-1-AP) [56]. Here input is two disjoint sets X, Y ($2|X| = |Y| = 2n$) together with costs a_{ijk} for all triples $(i, j, k) \in X \times Y \times Y$, and the goal is to find a set M of n mutually disjoint triples that minimize $\sum_{(i,j,k)\in M} a_{ijk}$. The 2-1-AP is NP-hard, even if costs a_{ijk} can have only two distinct values. Namely, it contains 3DA as a special case: Starting from an 3DA instance $I' = (X', Y', Z', \{a'_{ijk}\})$ where $a'_{ijk} \in \{u, v\}$, $u < v$, one can obtain an equivalent instance of the 2-1-AP $I = (X, Y, \{a_{ijk}\})$ by setting $X := X'$, $Y := Y' \cup Z'$ and defining costs a_{ijk} to be equal to corresponding a'_{ijk}, extended by setting the costs of new feasible triples to be equal to v.

Together with approximability analysis authors [56] suggest several practical problems that can be modeled with 2-1-AP, such as satellite refuelling, chromosome pairing, sports scheduling and gender matching. Main results are concerned about special case of 2-1-AP. Namely, when the input costs are decomposable (1.42) and satisfy triangle inequality there can not be a polynomial time approximation scheme, unless P=NP. Finally, paper presents 4/3-approximation for this version of 2-1-AP, which generalizes the similar result for 3AP.

General multi-dimensional assignment problems

LAP (2 dimensions) and axial 3AP (3 dimensions) could be easily generalized into more dimensions. Namely, *m-dimensional assignment problem* is to minimize

$$\sum_{i_1=1}^{n} \sum_{i_2=1}^{n} \cdots \sum_{i_m=1}^{n} a_{i_1 i_2 \dots i_m} x_{i_1 i_2 \dots i_m} \tag{1.53}$$

$$\text{subject to} \quad \sum_{i_2=1}^{n} \sum_{i_3=1}^{n} \cdots \sum_{i_m=1}^{n} x_{i_1 i_2 \dots i_m} = 1 \qquad (i_1 = 1, 2, \dots, n), \tag{1.54}$$

$$\sum_{i_1=1}^{n} \sum_{i_3=1}^{n} \cdots \sum_{i_m=1}^{n} x_{i_1 i_2 \dots i_m} = 1 \qquad (i_2 = 1, 2, \dots, n), \tag{1.55}$$

$$\vdots$$

$$\sum_{i_1=1}^{n} \sum_{i_2=1}^{n} \cdots \sum_{i_{m-1}=1}^{n} x_{i_1 i_2 \dots i_m} = 1 \qquad (i_m = 1, 2, \dots, n), \tag{1.56}$$

$$x_{i_1 i_2 \dots i_m} \in \{0, 1\} \qquad (i_1, i_2, \dots, i_m = 1, \dots, n). \tag{1.57}$$

Alternatively, you can state the problem as finding $m - 1$ permutations that solve

$$\min_{\pi_j \in \Pi, \forall j \in \{1, \dots, m-1\}} \sum_{i=1}^{n} a_{i\, \pi_1(i)\, \pi_2(i) \dots \pi_{m-1}(i)}. \tag{1.58}$$

m-dimensional assignment problems most commonly appear during modelling of multi-target tracking, routing in meshes and data association problems. By exchanging (1.54) - (1.56) with a larger number of constraints, each with more indices fixed (and so with less number of summations) one can define *planar m-dimensional assignment problem*. The connection of the solution structure to Latin squares is similar as in planar 3AP. This variant of the problem is useful for the design of tournaments, conflict-free access to parallel memories and the design of error-correcting codes. For more details and references on applications of multi-dimensional assignment problems see surveys [19, 122].

The dimensionality of the polytop for the m-dimensional assignment problem can be computed as $\sum_{i=0}^{m-2}\binom{m}{i}(n-1)^{m-i}$ due to result in [8].

Clearly, the problem is NP-hard for $m \geq 3$. Two papers [52, 88] discuss approximation algorithms and prove that they are asymptotically optimal for special cases of random input matrices. In case of maximizing the objective value $(1/m)$-approximation is presented in [72] and $(2/m-\epsilon)$-approximation in [73], with the restriction of $0, 1$ cost coefficients for the later. Moreover, for the $0, 1$ costs case, authors of [91] show that the ratio between optimal solution value and LP relaxation is bounded by $m-1$.

Special case of m-dimensional assignment problem with cost array satisfying Monge property (similar to 3AP description) admits trivial solution [22]. If cost coefficients are sum decomposable (that is $a_{i_1 i_2 \ldots i_m} = d_{i_1 i_2} + d_{i_1 i_3} + \ldots + d_{i_1 i_m} + d_{i_2 i_3} + \ldots + d_{i_{m-1} i_m}$), then there exists $(2-2/m)$-approximation algorithm. Proofs of approximation ratios for this and several others forms of decomposable cost coefficients are presented in [11]. A characterization of instances that have every solution with the same objective value for axial and planar m-dimensional assignment problems can be found in [80] and [34] respectively.

Authors in [62] consider the m-dimensional assignment problem with random cost coefficients and bound the expected number of local optimums for the 2-exchange search neighborhood and its generalizations.

Most of the algorithmic results on these general problems are based on Lagrangean relaxations and branch-and-cut. With the later, polynomial-time separation algorithm have been proposed for cuts based on clique, odd-hole, and antiweb inequalities. The only known metaheuristic developed for the problem is Greedy Randomized Adaptive Search with both serial and parallel adaptations. See [19] for references to these algorithms and to the analysis of probabilistic asymptotic behavior and probabilistic bounds on optimal solution value.

1.2.4 Partition assignment problem

Let $N = \{1, 2, \ldots, n\}$ and $E = \{1, 2, \ldots, p\}$ be finite sets and $F = \{S_1, S_2, \ldots, S_m\}$ be a family of subsets of E. The index set of elements of F is denoted by $M = \{1, 2, \ldots, m\}$. For each $(i, j) \in N \times M$, a cost c_{ij} is prescribed. The value c_{ij} can be viewed as the cost of assigning the set S_j to i. Then the *partition assignment problem* (PAP) is to find a partition $\{S_{k_1}, S_{k_2}, \ldots, S_{k_r}\}$ of E such that $S_{k_i} \in F$ for $i = 1, 2, \ldots, r$ and an assignment of $S_{k_1}, S_{k_2}, \ldots, S_{k_r}$ to $\{1, 2, \ldots, n\}$ such that the total cost of assignment is as small as possible.

An integer programming formulation of this problem can be given as follows. For each $t \in E$ let $\Delta(t) = \{j : t \in S_j\}$. Consider the decision variables x_{ij} given by

$$x_{ij} = \begin{cases} 1 & \text{if } i \in N \text{ is assigned to the subset } S_j, \\ 0 & \text{otherwise.} \end{cases}$$

Then PAP can be formulated as a 0-1 integer programming problem:

$$\begin{aligned} \text{Minimize} \quad & \sum_{i \in N} \sum_{j \in M} c_{ij} x_{ij} \\ \text{Subject to} \quad & \sum_{i \in N} x_{ij} \le 1, \quad \forall j \in M && (1.59) \\ & \sum_{j \in M} x_{ij} \le 1, \quad \forall i \in N && (1.60) \\ & \sum_{j \in \Delta(t)} \sum_{i \in N} x_{ij} = 1, \quad \forall t \in E && (1.61) \\ & x_{ij} \in \{0, 1\}, \quad \forall i \in N, \forall j \in M && (1.62) \end{aligned}$$

Note that in presence of equation (1.61), equation (1.59) is redundant and can be discarded. It may be advantageous to retain equation (1.59) in the model since it might provide a tighter continuous relaxation.

PAP can also be viewed as a bilevel program combining set partitioning and weighted bipartite matching as follows. Let $S^\alpha = \{S_{\alpha_1}, S_{\alpha_2}, \ldots, S_{\alpha_u}\}$ be a subfamily of F, such that S^α is a partition of E. Construct the complete bipartite graph $G^\alpha = (V^\alpha, H^\alpha)$ with the generic bipartition of $V^\alpha = N \cup U^\alpha$, where $U^\alpha = \{\alpha_1, \alpha_2, \ldots, \alpha_u\}$ and the cost of the edge $(i, \alpha_j) = c_{i\alpha_j}$. A matching in G^α is called *pseudo-perfect* if all nodes in U^α are matched. Let M^α be a minimum cost pseudo-perfect matching in G^α. Then the cost of the partition S^α is $C(S^\alpha) = \sum_{(i,j) \in M^\alpha} c_{ij}$. Note that $C(S^\alpha)$ is precisely the cost of the minimum cost pseudo-perfect matching in G^α. Thus PAP can be written as

$$\begin{aligned} &\text{Minimize} \quad C(S^\alpha) \\ &\text{Subject to } S^\alpha \in F' \end{aligned}$$

where F' is the collection of all partitions of E that are in F, that is, $F' = \{S^\alpha = \{S_{\alpha_1}, S_{\alpha_2}, \ldots, S_{\alpha_u}\} : S_{\alpha_i} \in F, 1 \le i \le u \le n, \text{ and } S^\alpha \text{ is a partition of } E\}$.

1.3 Outline of thesis

In this thesis we discuss our recent findings for the bilinear assignment problem with the focus on computational complexity, solvable special cases, approximations, linearizations, local search methods and related heuristics. We also talk about our results with several applied problems, where modelling with nonlinear assignment problems was instrumental.

An outline of the write-up is as follows. Chapter 2 is dedicated to theoretical results on the bilinear assignment problem. First we discuss the issues of the complexity of BAP and its special cases, and mention results of approximation and domination analysis. We also present several reformulation linearizations techniques to reformulate the problem as an integer linear program. Then, in Chapter 3 we discuss the key component in devising local search algorithms for the problem - search neighborhood structures. This is followed by the overview of our experimental experience with various local searches and more advanced heuristics for BAP. In Chapter 4 we present a ridesharing problem and several ways to model it. One of the possible models is a new variation of nonlinear assignment problem - partition assignment problem. We mention some preliminary results on this model. The next chapter (Chapter 5) is dedicated to another optimization problem, which was encountered during our work on scheduling deliveries for vehicles that carry laboratory samples. The summary of our modelling and computational experience with multi-commodity network flow formulation for the problem is provided. We summarize results of the thesis and mention key open problems in Chapter 6.

Chapter 2

Bilinear Assignment Problem: Complexity and Polynomially Solvable Special Cases

In this chapter we discuss our theoretical results on bilinear assignment problem published in Mathematical Programming journal from November 2017 [38].

We show that Bilinear Assignment Problem (BAP) cannot be approximated within a constant factor unless P=NP even if the associated quadratic cost matrix Q is diagonal. Further, we show that BAP remains NP-hard if $m = O(\sqrt[r]{n})$, for some fixed r, but is solvable in polynomial time if $m = O(\frac{\log n}{\log \log n})$. When the rank of Q is fixed, BAP is observed to admit FPTAS and when this rank is one, it is solvable in polynomial time under some additional restrictions. We then provide a necessary and sufficient condition for BAP to be equivalent to two linear assignment problems. A closed form expression to compute the average of the objective function values of all solutions is presented, whereas the median of the solution values cannot be identified in polynomial time, unless P=NP. We then provide polynomial time heuristic algorithms that find a solution with objective function value no worse than that of $(m-1)!(n-1)!$ solutions. However, computing a solution whose objective function value is no worse than that of $m!n! - \lceil \frac{m}{\beta} \rceil ! \lceil \frac{n}{\beta} \rceil !$ solutions is NP-hard for any fixed rational number $\beta > 1$.

This chapter is organized as follows. In Section 2.2 we investigate the complexity of BAP and prove that it is NP-hard if $m = O(\sqrt[r]{n})$, for some fixed r, but is polynomially solvable if $m = O(\frac{\log n}{\log \log n})$. Note that QAP is polynomially solvable if Q' is diagonal, but we show that BAP is NP-hard even if Q is diagonal and $n = m$. Moreover, such BAP instances do not admit a polynomial time α-approximation algorithm for any fixed $\alpha > 1$, unless P=NP. Section 2.2 also deals with some special classes of BAP that are solvable in polynomial time. In particular, we provide an algorithm to solve BAP when Q, observed as a matrix, is of rank one, and either C or D is a sum matrix (i.e., either c_{ij}'s or d_{ij}'s are of the form $s_i + t_j$, for some vectors $S = (s_i)$ and $T = (t_i)$). Furthermore, we show that a BAP instance is equivalent to two linear assignment problems, if and only if Q is of the form $q_{ijk\ell} = e_{ijk} + f_{ij\ell} + g_{ik\ell} + h_{jk\ell}$, for a natural definition of equivalency. In this case, the problem can be solved in $O(n^3)$ time. In Section 2.3, we analyze various methods for approximating an optimal solution. We show that there is a FPTAS for BAP if the rank of Q is fixed, and present a discretization procedure that preserves the objective value. Computing a solution whose objective function value is no worse than that of $m!n! - \lceil \frac{m}{\beta} \rceil ! \lceil \frac{n}{\beta} \rceil !$ solutions is observed to be NP-hard for any fixed rational number $\beta > 1$. We also obtain a closed form expression for computing the average of the objective function values of all feasible solutions, and present methods for finding a solution with the objective value guaranteed to be no worse than the average value. Such solutions are shown to have objective function value no worse than $(m-1)!(n-1)!$ alternative solutions. Finally, we note that a median solution cannot be found in polynomial time, unless P=NP. Here the median denotes a solution which is in the middle of the list of all feasible solutions sorted by their objective function value. Concluding remarks are given in Section 2.5.

2.1 Formulation and connection to other nonlinear problems

Let $Q = (q_{ijk\ell})$ be an $m \times m \times n \times n$ array, $C = (c_{ij})$ be an $m \times m$ matrix, and $D = (d_{k\ell})$ be an $n \times n$ matrix. Then the *bilinear assignment problem* (BAP) is to

$$
\begin{aligned}
\text{Minimize} \quad & \sum_{i=1}^{m}\sum_{j=1}^{m}\sum_{k=1}^{n}\sum_{\ell=1}^{n} q_{ijk\ell} x_{ij} y_{k\ell} + \sum_{i=1}^{m}\sum_{j=1}^{m} c_{ij} x_{ij} + \sum_{k=1}^{n}\sum_{\ell=1}^{n} d_{k\ell} y_{k\ell} && (2.1)\\
\text{subject to} \quad & \sum_{j=1}^{m} x_{ij} = 1 \qquad i = 1, 2, \ldots, m, && (2.2)\\
& \sum_{i=1}^{m} x_{ij} = 1 \qquad j = 1, 2, \ldots, m, && (2.3)\\
& \sum_{\ell=1}^{n} y_{k\ell} = 1 \qquad k = 1, 2, \ldots, n, && (2.4)\\
& \sum_{k=1}^{n} y_{k\ell} = 1 \qquad \ell = 1, 2, \ldots, n, && (2.5)\\
& x_{ij},\ y_{k\ell} \in \{0, 1\} \qquad i, j = 1, \ldots, m, \quad k, \ell = 1, \ldots, n. && (2.6)
\end{aligned}
$$

Let $\mathcal{X}$ be the set of all $m \times m$ 0-1 matrices satisfying (2.2) and (2.3) and $\mathcal{Y}$ be the set of all $n \times n$ 0-1 matrices satisfying (2.4) and (2.5). Also, let $\mathcal{F}$ be the set of all feasible solutions of BAP. Note that $|\mathcal{F}| = m!n!$. An instance of the BAP is fully defined by the 3-tuple of cost arrays (Q, C, D). Let $M = \{1, 2, \ldots, m\}$ and $N = \{1, 2, \ldots, n\}$. Without loss of generality we assume that $m \leq n$. The objective function of BAP is denoted by

$f(\mathbf{x}, \mathbf{y})$ where $\mathbf{x} = (x_{ij}) \in \mathcal{X}$ and $\mathbf{y} = (y_{ij}) \in \mathcal{Y}$. The quadratic part of the objective function, i.e. $\sum_{i=1}^{m} \sum_{j=1}^{m} \sum_{k=1}^{n} \sum_{\ell=1}^{n} q_{ijk\ell} x_{ij} y_{k\ell}$, is denoted by $\bar{f}(\mathbf{x}, \mathbf{y})$. It may be noted that in BAP, constraints (2.6) can be replaced by $0 \leq x_{ij} \leq 1$ and $0 \leq y_{k\ell} \leq 1$ for $i, j = 1, \ldots, m$, and $k, \ell = 1, \ldots, n$. This is because, there always exist an integral optimal solution of the relaxed version of the problem, which can be shown by applying the analogous property for the linear assignment problem twice, e.g. see Theorem 10. This justifies the name bilinear assignment problem [5, 85].

BAP is closely related to the well known *quadratic assignment problem* (QAP) [25]. Let $Q' = (q'_{ijk\ell})$ be an $n \times n \times n \times n$ array. Then the QAP is defined as

$$\begin{aligned}
\text{Minimize} \quad & \sum_{i=1}^{n} \sum_{j=1}^{n} \sum_{k=1}^{n} \sum_{\ell=1}^{n} q'_{ijk\ell} x_{ij} x_{k\ell} \\
\text{subject to} \quad & \sum_{j=1}^{n} x_{ij} = 1 \qquad i = 1, 2, \ldots, n, \\
& \sum_{i=1}^{n} x_{ij} = 1 \qquad j = 1, 2, \ldots, n, \\
& x_{ij} \in \{0, 1\} \qquad i, j = 1, \ldots, n.
\end{aligned}$$

Note that Q' could be viewed as an $n^2 \times n^2$ matrix. Let $Q'' = Q' + \alpha I$ where I is the $n^2 \times n^2$ identity matrix. It is well known that the optimal solution set for the QAP with cost matrix Q' and Q'' are identical. By choosing α appropriately, we could make Q'' either positive semidefinite or negative semidefinite. Thus without loss of generality we could assume that Q' in QAP is symmetric and negative semidefinite [18, 25]. Under this assumption, if $(\mathbf{x}^*, \mathbf{y}^*)$ is an optimal solution to the BAP instance (Q', O^n, O^n), where O^n is an $n \times n$ zero matrix, then both $\mathbf{x}^*$ and $\mathbf{y}^*$ are optimal solutions to the QAP [39, 85]. Thus, if $\mathbf{x}^* \neq \mathbf{y}^*$, they are alternative optimal solutions for the QAP. In this sense, BAP is a generalization of the QAP. It may be noted that [85] considers maximization of a convex quadratic function with linear constraints and its linkages to solving a corresponding bilinear program. Our assumption that Q' in QAP is negative semidefinite leads to the question of minimization of a concave quadratic function and its connection to minimization of a bilinear program. The pertinent results in [85] extend in a natural way to minimization of a concave quadratic function and the minimization of the corresponding bilinear program. This establishes the validity of our remarks on BAP as a generalization of QAP.

Another interesting combinatorial optimization problem related to BAP, is the so called *independent QAP*, introduced in [21]. It can be reformulated as follows. Given two $n \times n$ matrices $A = (a_{ij})$ and $B = (b_{ij})$, one needs to find $\mathbf{x}, \mathbf{y} \in \mathcal{Y}$ such that

$$\sum_{i=1}^{n} \sum_{j=1}^{n} \sum_{k=1}^{n} \sum_{\ell=1}^{n} a_{ik} b_{j\ell} x_{ij} y_{k\ell}$$

is minimized. Then, it is clear that an instance of independent QAP can be reduced to an instance (Q, O^n, O^n) of BAP, where $q_{ijk\ell} = a_{ik}b_{j\ell}$. Therefore, independent QAP is a special case of BAP.

Furthermore, the *axial three-dimensional assignment problem* (3AP) [122] is also a special case of BAP. Consider the $n \times n \times n$ cost array $A = (a_{ijk})$. Then the 3AP is to find $\mathbf{x}, \mathbf{y} \in \mathcal{Y}$ such that

$$\sum_{i=1}^{n} \sum_{j=1}^{n} \sum_{k=1}^{n} a_{ijk} x_{ij} y_{jk}$$

is minimized. Now it is easy to see that every 3AP instance with $A = (a_{ijk})$ can be reduced to a BAP instance (Q, O^n, O^n) where $q_{ijk\ell}$ is equal to $a_{ij\ell}$ if $j = k$, and 0 otherwise. The above reduction of 3AP to BAP is a special case of a more general problem considered by Frieze [46].

2.2 Complexity and polynomially solvable cases

Since BAP is a generalization of QAP, it is clearly strongly NP-hard. It is well known that for any $\alpha > 1$ the existence of a polynomial time α-approximation algorithm for QAP implies P=NP [116]. The reduction from QAP to BAP discussed in the introduction section need not preserve approximation ratio. Let us now discuss another reduction from QAP to BAP that partially preserves approximation ratio.

Suppose $m = n$ and impose the additional restriction in BAP that $x_{ij} = y_{ij}$ for all i, j. The resulting problem is equivalent to QAP. Constraints $x_{ij} = y_{ij}$ can be enforced by modifying the entries of Q, C and D without explicitly stating the constraints or changing the objective function values of the solutions that satisfy the new constants. For every i, j we can change c_{ij} to $c_{ij} + L$, d_{ij} to $d_{ij} + L$ and q_{ijij} to $q_{ijij} - 2L$, for some large L, e.g. $L = m \max_{i,j} |c_{ij}| + n \max_{k,\ell} |d_{k\ell}| + mn \max_{i,j,k,\ell} |q_{ijk\ell}| + 1$. This will increase the objective value by $\sum_{i,j=1}^{n} L(x_{ij} - 2x_{ij}y_{ij} + y_{ij}) = \sum_{i,j=1}^{n} L(x_{ij} - y_{ij})^2$, which forces $x_{ij} = y_{ij}$ in an optimal solution. Since the reduction described above preserves the objective values of the solutions that satisfy $x_{ij} = y_{ij}$, BAP inherits the approximation hardness of QAP. That is, for any $\alpha > 1$, BAP does not have a polynomial time α-approximation algorithm, unless P=NP.

More interestingly we now show that this non-approximability result for BAP extends to the case even if Q is a diagonal matrix. Note that QAP is polynomially solvable when Q' is diagonal.

Theorem 1. *BAP is NP-hard even if Q is a diagonal matrix. Further, for any $\alpha > 1$ the existence of a polynomial time α-approximation algorithm for BAP when Q is a diagonal matrix implies P=NP.*

Proof. To prove the theorem, we reduce the DISJOINT MATCHINGS problem to BAP. Input of the problem is given by a complete bipartite graph $K_{n,n} = (V_1, V_2, E)$ and two sets of edges $E_1, E_2 \subseteq E$. The goal is to decide if there exist $M_1 \subseteq E_1$ and $M_2 \subseteq E_2$, such that M_1 and M_2 are perfect matchings and $M_1 \cap M_2 = \emptyset$. DISJOINT MATCHINGS is shown to be NP-complete by Frieze [47]. A simpler proof can be found in [45].

Now let us assume that for some $\alpha > 1$ there exists a polynomial time α-approximation algorithm for BAP when Q is a diagonal matrix. Consider an instance of DISJOINT MATCHINGS defined on $K_{n,n} = (V_1, V_2, E)$, the edge set E_1 and the edge set E_2, where $V_1 = \{v_1, v_2, \ldots, v_n\}$ and $V_2 = \{u_1, u_2, \ldots, u_n\}$. Let (Q, C, D) be an instance of BAP where the $n \times n \times n \times n$ array $Q = (q_{ijk\ell})$ is the identity matrix when observed as an $n^2 \times n^2$ matrix (i.e., $q_{ijk\ell} = 1$ if $i = k$, $j = \ell$, and 0 otherwise). Further, let $C = (c_{ij})$ and $D = (d_{ij})$ be $n \times n$ matrices given by

$$c_{ij} = \begin{cases} \frac{1}{\alpha+1} & \text{if } (v_i, u_j) \in E_1 \text{ and } i = 1, \\ 0 & \text{if } (v_i, u_j) \in E_1 \text{ and } i \neq 1, \\ 1 & \text{otherwise,} \end{cases} \quad \text{and} \quad d_{ij} = \begin{cases} 0 & \text{if } (v_i, u_j) \in E_2, \\ 1 & \text{otherwise.} \end{cases} \tag{2.7}$$

Now we show that our DISJOINT MATCHINGS instance is a "yes" instance, if and only if there exists a solution $(\mathbf{x}^*, \mathbf{y}^*)$ of BAP on the instance (Q, C, D) with the objective value $\frac{1}{\alpha+1}$. Assume that $(\mathbf{x}^*, \mathbf{y}^*) \in \mathcal{F}$ is such that $f(\mathbf{x}^*, \mathbf{y}^*) = \frac{1}{\alpha+1}$. Since all costs are non-negative, and $c_{1j} \geq \frac{1}{\alpha+1}$ for all $j = 1, \ldots, n$, it follows that $\sum_{i=1}^n \sum_{j=1}^n c_{ij} x^*_{ij} = \frac{1}{\alpha+1}$, which implies that $\mathbf{x}^*$ corresponds to a perfect matching S_1 which is a subset of E_1. Similarly, it must be that $\sum_{i=1}^n \sum_{j=1}^n d_{ij} y^*_{ij} = 0$, hence $\mathbf{y}^*$ corresponds to a perfect matching S_2 which is a subset of E_2. Furthermore, $f(\mathbf{x}^*, \mathbf{y}^*) = \frac{1}{\alpha+1}$ implies that $\sum_{i=1}^n \sum_{j=1}^n \sum_{k=1}^n \sum_{\ell=1}^n q_{ijk\ell} x^*_{ij} y^*_{k\ell} = 0$. Then since Q is an identity matrix, it follows that S_1 and S_2 are disjoint, hence our DISJOINT MATCHINGS instance is a "yes" instance. Conversely, if we assume that our DISJOINT MATCHINGS instance is a "yes" instance, then by the same arguments but in opposite direction, it follows that an optimal solution of the BAP instance (Q, C, D) has the objective value equal to $\frac{1}{\alpha+1}$.

For every BAP instance (Q, C, D) obtained from a DISJOINT MATCHINGS instance using the reduction described above, if the objective value of an arbitrary feasible solution is not $\frac{1}{\alpha+1}$, it will be at least 1. Since $\alpha \frac{1}{\alpha+1} < 1$, our α-approximation algorithm for the BAP with diagonal matrices Q can be used for solving the NP-complete DISJOINT MATCHINGS problem, which implies that P=NP. □ □

If we replace costs $\frac{1}{\alpha+1}$ in (2.7) with 0, we get a reduction from DISJOINT MATCHINGS to BAP instances with 0-1 costs. Therefore, BAP is NP-hard even when restricted to instances with 0-1 costs where Q is the identity matrix.

Let us now examine the impact of the ratio between m and n on the tractability of BAP.

Theorem 2. *If $m = O(\frac{\log n}{\log\log n})$ then BAP can be solved in polynomial time. However, for any fixed r, if $m = O(\sqrt[r]{n})$, then BAP is NP-hard.*

Proof. For a given $\mathbf{x}^* \in \mathcal{X}$, one can find a $\mathbf{y} \in \mathcal{Y}$ which minimizes $f(\mathbf{x}^*, \mathbf{y})$ by investing $O(n^3)$ time. This is achieved by solving a *linear assignment problem* (LAP) [18] with cost matrix $H = (h_{k\ell})$ where $h_{k\ell} = d_{k\ell} + \sum_{i=1}^{m}\sum_{j=1}^{m} q_{ijk\ell}x^*_{ij}$. Hence, if $|\mathcal{X}| = m!$ is a polynomial in n, then BAP can be solved in polynomial time by enumerating $\mathcal{X}$ and solving all corresponding LAPs. Assume that $m = O(\frac{\log n}{\log\log n})$, i.e., $m \le \frac{a\log n}{\log\log n}$ for some constant $a \ge 1$. Then, using the fact that $\log(m!) \le m\log m$, we have

$$\log(m!) \le a\log n + \frac{a\log a\log n}{\log\log n} - \frac{a\log n\log\log\log n}{\log\log n} \le (a + a\log a)\log n.$$

Thus $|\mathcal{X}| = m! \le n^{a+a\log a}$, establishing the first part of the theorem.

Now we prove the second part. Let (Q, C, D) be an arbitrary instance of BAP with size parameters m and n. Given a constant r, consider the $n \times n \times n^r \times n^r$ cost array $\hat{Q}$, the $n \times n$ matrix $\hat{C}$ and the $n^r \times n^r$ matrix $\hat{D}$ defined as follows:

$$\hat{q}_{ijk\ell} = \begin{cases} q_{ijk\ell} & \text{if } i,j \le m \text{ and } k,\ell \le n,\\ 0 & \text{otherwise,}\end{cases} \qquad \hat{c}_{ij} = \begin{cases} c_{ij} & \text{if } i,j \le m,\\ 0 & \text{if } i,j > m,\\ L & \text{otherwise,}\end{cases} \qquad \hat{d}_{k\ell} = \begin{cases} d_{k\ell} & \text{if } k,\ell \le n,\\ 0 & \text{if } k,\ell > n,\\ L & \text{otherwise,}\end{cases}$$

where L is a large number. It is easy to see that the BAP instances (Q, C, D) and $(\hat{Q}, \hat{C}, \hat{D})$ are equivalent in the sense that from an optimal solution to one problem, an optimal solution to other can be recovered in polynomial time. $(\hat{Q}, \hat{C}, \hat{D})$ satisfies the condition of the second statement of the theorem and it can be constructed in polynomial time, therefore the result follows. □ □

By analogous arguments as in the proof above, we get that if $m = O(\frac{\log^a n}{\log\log n})$, for some constant a, then BAP is quasi-polynomial time solvable.

Polynomially solvable special cases of the QAP and the 3AP have been investigated by many researchers [18, 122]. Since QAP and 3AP are special cases of the BAP, all of their polynomially solvable special cases can be mapped onto polynomially solvable special cases for the BAP. Let us now consider some new polynomially solvable cases.

Note that the rows of Q can be labeled using the ordered pairs (i, j), $i, j = 1, 2, \ldots, m$. Then the row of Q represented by the label (i, j) can be viewed as an $n \times n$ matrix $P^{ij} = (p^{ij}_{k\ell})$ where $p^{ij}_{k\ell} = q_{ijk\ell}$ for $k, \ell = 1, 2, \ldots, n$. A linear assignment problem (LAP) with cost matrix P^{ij} has *constant value property* (CVP) if there exists a constant α_{ij} such that $\sum_{k=1}^{n}\sum_{\ell=1}^{n} p^{ij}_{k\ell}y_{k\ell} = \alpha_{ij}$ for all $\mathbf{y} \in \mathcal{Y}$. Characterizations of cost matrices with CVP for various combinatorial optimization problems have been studied by different authors [12, 34].

In the case when LAP with cost matrix P^{ij} has CVP for all $i, j = 1, 2, \dots, m$, we define $W = w_{ij}$ to be an $m \times m$ matrix where $w_{ij} = \alpha_{ij} + c_{ij}$.

Theorem 3. *If P^{ij} satisfies CVP for all $i, j = 1, 2, \dots, m$, $\mathbf{x}^*$ is an optimal solution to the LAP with cost matrix W, and $\mathbf{y}^*$ is an optimal solution to the LAP with cost matrix D, then $(\mathbf{x}^*, \mathbf{y}^*)$ is an optimal solution to the BAP (Q, C, D).*

Proof.

$$\begin{aligned}
\min_{\mathbf{x}\in\mathcal{X},\mathbf{y}\in\mathcal{Y}} f(\mathbf{x},\mathbf{y}) &= \min_{\mathbf{x}\in\mathcal{X},\mathbf{y}\in\mathcal{Y}} \left\{ \sum_{i=1}^{m}\sum_{j=1}^{m}\sum_{k=1}^{n}\sum_{\ell=1}^{n} q_{ijk\ell}x_{ij}y_{k\ell} + \sum_{i=1}^{m}\sum_{j=1}^{m} c_{ij}x_{ij} + \sum_{k=1}^{n}\sum_{\ell=1}^{n} d_{k\ell}y_{k\ell} \right\} \\
&= \min_{\mathbf{x}\in\mathcal{X},\mathbf{y}\in\mathcal{Y}} \left\{ \sum_{i=1}^{m}\sum_{j=1}^{m} \alpha_{ij}x_{ij} + \sum_{i=1}^{m}\sum_{j=1}^{m} c_{ij}x_{ij} + \sum_{k=1}^{n}\sum_{\ell=1}^{n} d_{k\ell}y_{k\ell} \right\} \\
&= \min_{\mathbf{x}\in\mathcal{X}} \left\{ \sum_{i=1}^{m}\sum_{j=1}^{m} (\alpha_{ij} + c_{ij})x_{ij} \right\} + \min_{\mathbf{y}\in\mathcal{Y}} \left\{ \sum_{k=1}^{n}\sum_{\ell=1}^{n} d_{k\ell}y_{k\ell} \right\}
\end{aligned}$$

Thus, BAP decomposes into two LAPs and the result follows. □ □

An analogous result can be derived using CVP for columns of Q. We omit the details.

A linear assignment problem (LAP) with cost matrix P^{ij} has *two value property* (2VP) if there exist some constants α_{ij} and β_{ij} such that $\sum_{k=1}^{n}\sum_{\ell=1}^{n} p_{k\ell}^{ij} y_{k\ell}$ is equal to either α_{ij} or β_{ij}, for all $\mathbf{y} \in \mathcal{Y}$. Tarasov [125] gave a characterization of cost matrices for LAP having 2VP. In view of Theorem 3, it would be interesting to explore the complexity of BAP when P^{ij} satisfies 2VP, for all i, j. As shown below, the polynomial solvability of BAP does not extend if CVP of matrices P^{ij} is replaced by 2VP.

Theorem 4. *BAP remains strongly NP-hard even if P^{ij} satisfies 2VP for all $i, j = 1, 2, \dots, m$.*

Proof. If Q is a diagonal matrix with no zero entries on the diagonal, then it is easy to verify that P^{ij} satisfies 2VP for all $i, j = 1, 2, \dots, m$. The result now follows from Theorem 1. □

□

2.2.1 Characterization of linearizable instances

In Theorem 3 we observed that there are special cases of BAP that can be solved by solving two independent LAPs. We generalize this by characterizing instances of BAP that are 'equivalent' to two independent LAPs. Let us first introduce some definitions.

An $m \times m \times n \times n$ cost array $Q = (q_{ijk\ell})$ associated with a BAP is said to be *linearizable* if there exist an $m \times m$ matrix $A = (a_{ij})$ and an $n \times n$ matrix $B = (b_{ij})$, such that

$$\sum_{i=1}^{m}\sum_{j=1}^{m}\sum_{k=1}^{n}\sum_{\ell=1}^{n} q_{ijk\ell}x_{ij}y_{k\ell} = \sum_{i=1}^{m}\sum_{j=1}^{m} a_{ij}x_{ij} + \sum_{k=1}^{n}\sum_{\ell=1}^{n} b_{k\ell}y_{k\ell}$$

for every $\mathbf{x} \in \mathcal{X}$ and $\mathbf{y} \in \mathcal{Y}$. The matrices A and B are called the linearization of Q.

Note that the collection of all linearizable $m \times m \times n \times n$ cost arrays forms a subspace of $\mathbb{R}^{m^2 \times n^2}$. If Q is linearizable, and a linearization (i.e. matrices A and B) is given, then the BAP instance (Q, C, D) is solvable in $O(m^3 + n^3)$ time for every C and D. I.e., such BAP instances reduce to two LAPs with respective cost matrices $A + C$ and $B + D$. Therefore we say that an instance (Q, C, D) of BAP is linearizable if and only if Q is linearizable. Linearizable instances of QAP have been studied by various authors [74, 107, 27]. Corresponding properties for the *quadratic spanning tree problem* (QMST) was investigated in [37]. Before attempting to characterize linearizable instances of BAP, let us first establish some preliminary results.

Lemma 5. *If the cost array $Q = (q_{ijk\ell})$ of a BAP satisfies*

$$q_{ijk\ell} = e_{ijk} + f_{ij\ell} + g_{ik\ell} + h_{jk\ell} \qquad i, j \in N,\ k, \ell \in M, \tag{2.8}$$

for some $m \times m \times n$ arrays $E = (e_{ijk})$, $F = (f_{ijk})$ and $m \times n \times n$ arrays $G = (g_{ijk})$, $H = (h_{ijk})$, then Q is linearizable.

Proof. Let Q be of the form (2.8), for some $E = (e_{ijk})$, $F = (f_{ijk})$, $G = (g_{ijk})$ and $H = (h_{ijk})$. Then for every $\mathbf{x} \in \mathcal{X}$ and $\mathbf{y} \in \mathcal{Y}$ we have that

$$\begin{aligned}
\bar{f}(\mathbf{x}, \mathbf{y}) &= \sum_{i=1}^{m} \sum_{j=1}^{m} \sum_{k=1}^{n} \sum_{\ell=1}^{n} q_{ijk\ell} x_{ij} y_{k\ell} = \sum_{i=1}^{m} \sum_{j=1}^{m} \sum_{k=1}^{n} \sum_{\ell=1}^{n} \left(e_{ijk} + f_{ij\ell} + g_{ik\ell} + h_{jk\ell}\right) x_{ij} y_{k\ell} \\
&= \sum_{i=1}^{m} \sum_{j=1}^{m} \sum_{k=1}^{n} \sum_{\ell=1}^{n} \left(e_{ijk} + f_{ij\ell}\right) x_{ij} y_{k\ell} + \sum_{i=1}^{m} \sum_{j=1}^{m} \sum_{k=1}^{n} \sum_{\ell=1}^{n} \left(g_{ik\ell} + h_{jk\ell}\right) x_{ij} y_{k\ell} \\
&= \sum_{i=1}^{m} \sum_{j=1}^{m} \left(\sum_{k=1}^{n} \sum_{\ell=1}^{n} \left(e_{ijk} + f_{ij\ell}\right) y_{k\ell} \right) x_{ij} + \sum_{k=1}^{n} \sum_{\ell=1}^{n} \left(\sum_{i=1}^{m} \sum_{j=1}^{m} \left(g_{ik\ell} + h_{jk\ell}\right) x_{ij} \right) y_{k\ell}.
\end{aligned} \tag{2.9}$$

Note that LAP given on a sum matrix T (i.e. of the form $t_{ij} = r_i + s_j$) has CVP. Moreover, the constant objective function value for such matrix is precisely $\sum_i r_i + \sum_j s_j$. Hence, for every $\mathbf{y} \in \mathcal{Y}$ and every $\mathbf{x} \in \mathcal{X}$

$$\sum_{k=1}^{n} \sum_{\ell=1}^{n} \left(e_{ijk} + f_{ij\ell}\right) y_{k\ell} = \sum_{k=1}^{n} e_{ijk} + \sum_{\ell=1}^{n} f_{ij\ell} = a_{ij} \text{ (say)},$$

and

$$\sum_{i=1}^{m} \sum_{j=1}^{m} \left(g_{ik\ell} + h_{jk\ell}\right) x_{ij} = \sum_{i=1}^{m} g_{ik\ell} + \sum_{j=1}^{m} h_{jk\ell} = b_{kl} \text{ (say)}.$$

Then, from (2.9) we have,

$$\sum_{i=1}^{m}\sum_{j=1}^{m}\sum_{k=1}^{n}\sum_{\ell=1}^{n} q_{ijk\ell}x_{ij}y_{k\ell} = \sum_{i=1}^{m}\sum_{j=1}^{m} a_{ij}x_{ij} + \sum_{k=1}^{n}\sum_{\ell=1}^{n} b_{k\ell}y_{k\ell},$$

which proves the lemma. $\square$ $\square$

Later we will show that the sufficient condition (2.8) is also necessary for Q to be linearizable. Arrays satisfying (2.8) are called *sum decomposable arrays with parameters* 4 *and* 3 in [34], where general sum decomposable arrays were investigated.

To establish that condition (2.8) is also necessary for linearizability of the associated BAP, we use the following lemma.

Lemma 6. *If the cost array $Q = (q_{ijk\ell})$ associated with a BAP satisfies*

$$\bar{f}(\mathbf{x},\mathbf{y}) = \sum_{i=1}^{m}\sum_{j=1}^{m}\sum_{k=1}^{n}\sum_{\ell=1}^{n} q_{ijk\ell}x_{ij}y_{k\ell} = K \tag{2.10}$$

for all $\mathbf{x} \in \mathcal{X}$, $\mathbf{y} \in \mathcal{Y}$ and some constant K, then Q must be of the form (2.8)*.*

Proof. Let $i, j \in M$ and $k, \ell \in N$ be chosen arbitrary such that $i, j, k, \ell \geq 2$. Also, let $\mathbf{x}^1, \mathbf{x}^2 \in \mathcal{X}$ be such that they only differ on index pairs from $\{1, i\} \times \{1, j\}$. In particular, let $x^1_{11} = x^1_{ij} = x^2_{1j} = x^2_{i1} = 1$. Similarly, let $\mathbf{y}^1, \mathbf{y}^2 \in \mathcal{Y}$ be such that they only differ on index pairs from $\{1, k\} \times \{1, \ell\}$ and in particular, let $y^1_{11} = y^1_{k\ell} = y^2_{1\ell} = y^2_{k1} = 1$.

Now let us assume that (2.10) is true for every $\mathbf{x} \in \mathcal{X}$ and $\mathbf{y} \in \mathcal{Y}$. Then it follows that

$$\bar{f}(\mathbf{x}^1,\mathbf{y}^1) + \bar{f}(\mathbf{x}^2,\mathbf{y}^2) = \bar{f}(\mathbf{x}^1,\mathbf{y}^2) + \bar{f}(\mathbf{x}^2,\mathbf{y}^1). \tag{2.11}$$

If we expand objective value expressions in (2.11) and cancel out identical parts from the left and right sides, we get

$$\begin{aligned} q_{1111} + q_{11k\ell} + q_{ij11} + q_{ijk\ell} + q_{1j1\ell} + q_{1jk1} + q_{i11\ell} + q_{i1k1} = \\ = q_{111\ell} + q_{11k1} + q_{ij1\ell} + q_{ijk1} + q_{1j11} + q_{1jk\ell} + q_{i111} + q_{i1k\ell}, \end{aligned}$$

from which it follows that

$$\begin{aligned} q_{ijk\ell} = \quad & q_{ijk1} + q_{ij1\ell} + q_{i1k\ell} + q_{1jk\ell} \\ & - q_{ij11} - q_{i1k1} - q_{i11\ell} - q_{1jk1} - q_{1j1\ell} - q_{11k\ell} \\ & + q_{i111} + q_{1j11} + q_{11k1} + q_{111\ell} \\ & - q_{1111} \end{aligned} \tag{2.12}$$

for every $i, j, k, \ell \geq 2$. However, note that (2.12) also holds true when some of i, j, k, ℓ are equal to 1. Namely, if we replace one of i, j, k, ℓ with 1, everything cancels out. Hence (2.12) holds for all $i, j \in M,\ k, \ell \in N$.

Define the $m \times m \times n$ arrays $A = (a_{ijk})$, $B = (b_{ijk})$ and the $m \times n \times n$ arrays $C = (c_{ijk})$, $D = (d_{ijk})$ as

$$\begin{aligned}
a_{ijk} &= q_{ijk1} - \frac{1}{2}q_{ij11} - \frac{1}{2}q_{i1k1} - \frac{1}{2}q_{1jk1} + \frac{1}{3}q_{i111} + \frac{1}{3}q_{1j11} + \frac{1}{3}q_{11k1} - \frac{1}{4}q_{1111},\\
b_{ij\ell} &= q_{ij1\ell} - \frac{1}{2}q_{ij11} - \frac{1}{2}q_{i11\ell} - \frac{1}{2}q_{1j1\ell} + \frac{1}{3}q_{i111} + \frac{1}{3}q_{1j11} + \frac{1}{3}q_{111\ell} - \frac{1}{4}q_{1111},\\
c_{ik\ell} &= q_{i1k\ell} - \frac{1}{2}q_{i1k1} - \frac{1}{2}q_{i11\ell} - \frac{1}{2}q_{11k\ell} + \frac{1}{3}q_{i111} + \frac{1}{3}q_{11k1} + \frac{1}{3}q_{111\ell} - \frac{1}{4}q_{1111},\\
d_{jk\ell} &= q_{1jk\ell} - \frac{1}{2}q_{1jk1} - \frac{1}{2}q_{1j1\ell} - \frac{1}{2}q_{11k\ell} + \frac{1}{3}q_{1j11} + \frac{1}{3}q_{11k1} + \frac{1}{3}q_{111\ell} - \frac{1}{4}q_{1111}.
\end{aligned}$$

Then, from (2.12) if follows that

$$q_{ijk\ell} = a_{ijk} + b_{ij\ell} + c_{ik\ell} + d_{jk\ell},$$

which proves the lemma. □ □

We are now ready to prove the characterization of linearization instances of BAP.

Theorem 7. *The cost array $Q = (q_{ijk\ell})$ of a BAP is linearizable if and only if it is of the form*

$$q_{ijk\ell} = e_{ijk} + f_{ij\ell} + g_{ik\ell} + h_{jk\ell} \qquad i, j \in N,\ k, \ell \in M, \tag{2.13}$$

for some $m \times m \times n$ arrays $E = (e_{ijk})$, $F = (f_{ijk})$ and $m \times n \times n$ arrays $G = (g_{ijk})$, $H = (h_{ijk})$.

Proof. Lemma 5 tells us that (2.13) is a sufficient condition for Q to be linearizable. To show that this is also a necessary condition we start by following the steps of the proof of Lemma 5 in reverse.

Let Q be a linearizable cost array. That is, there exist some $A = (a_{ij})$ and $B = (b_{ij})$ such that

$$\bar{f}(\mathbf{x}, \mathbf{y}) = \sum_{i=1}^{m}\sum_{j=1}^{m}\sum_{k=1}^{n}\sum_{\ell=1}^{n} q_{ijk\ell}x_{ij}y_{k\ell} = \sum_{i=1}^{m}\sum_{j=1}^{m} a_{ij}x_{ij} + \sum_{k=1}^{n}\sum_{\ell=1}^{n} b_{k\ell}y_{k\ell} \tag{2.14}$$

for every $\mathbf{x} \in \mathcal{X}$ and $\mathbf{y} \in \mathcal{Y}$. For every $i, j \in M$ we can choose $2n$ numbers $\hat{e}_{ijk}$, $\hat{f}_{ijk}$, $k = 1, \ldots, n$, such that $\sum_{k=1}^{n}(\hat{e}_{ijk} + \hat{f}_{ijk}) = a_{ij}$. Furthermore, for every $k, \ell \in N$ we can choose $2m$ numbers $\hat{g}_{ik\ell}$, $\hat{h}_{ik\ell}$, $i = 1, \ldots, m$, such that $\sum_{i=1}^{m}(\hat{g}_{ik\ell} + \hat{h}_{ik\ell}) = b_{k\ell}$. Now (2.14) can be expressed as

$$\begin{aligned}
\bar{f}(\mathbf{x},\mathbf{y}) &= \sum_{i=1}^{m}\sum_{j=1}^{m}\left(\sum_{k=1}^{n}\hat{e}_{ijk}+\sum_{\ell=1}^{n}\hat{f}_{ij\ell}\right)x_{ij}+\sum_{k=1}^{n}\sum_{\ell=1}^{n}\left(\sum_{i=1}^{m}\hat{g}_{ik\ell}+\sum_{j=1}^{m}\hat{h}_{jk\ell}\right)y_{k\ell} \\
&= \sum_{i=1}^{m}\sum_{j=1}^{m}\left(\sum_{k=1}^{n}\sum_{\ell=1}^{n}\left(\hat{e}_{ijk}+\hat{f}_{ij\ell}\right)y_{k\ell}\right)x_{ij}+\sum_{k=1}^{n}\sum_{\ell=1}^{n}\left(\sum_{i=1}^{m}\sum_{j=1}^{m}\left(\hat{g}_{ik\ell}+\hat{h}_{jk\ell}\right)x_{ij}\right)y_{k\ell} \\
&= \sum_{i=1}^{m}\sum_{j=1}^{m}\sum_{k=1}^{n}\sum_{\ell=1}^{n}\left(\hat{e}_{ijk}+\hat{f}_{ij\ell}\right)x_{ij}y_{k\ell}+\sum_{i=1}^{m}\sum_{j=1}^{m}\sum_{k=1}^{n}\sum_{\ell=1}^{n}\left(\hat{g}_{ik\ell}+\hat{h}_{jk\ell}\right)x_{ij}y_{k\ell} \\
&= \sum_{i=1}^{m}\sum_{j=1}^{m}\sum_{k=1}^{n}\sum_{\ell=1}^{n}\left(\hat{e}_{ijk}+\hat{f}_{ij\ell}+\hat{g}_{ik\ell}+\hat{h}_{jk\ell}\right)x_{ij}y_{k\ell}. \qquad (2.15)
\end{aligned}$$

Expression (2.15) implies that

$$\sum_{i=1}^{m}\sum_{j=1}^{m}\sum_{k=1}^{n}\sum_{\ell=1}^{n}\left(q_{ijk\ell}-\left(\hat{e}_{ijk}+\hat{f}_{ij\ell}+\hat{g}_{ik\ell}+\hat{h}_{jk\ell}\right)\right)x_{ij}y_{k\ell}=0 \qquad \text{for all } \mathbf{x}\in\mathcal{X},\ \mathbf{y}\in\mathcal{Y}.$$

Now from Lemma 6, it follows that the $m\times m\times n\times n$ array with entries $q_{ijk\ell}-(\hat{e}_{ijk}+\hat{f}_{ij\ell}+\hat{g}_{ik\ell}+\hat{h}_{jk\ell})$ is of the form (2.13), which implies that Q itself is of the form (2.13). This concludes the proof. □ □

A natural question that arises is the following: Given a BAP cost array Q, can we determine whether Q is linearizable, and if so, how can we find its linearization matrices A and B? The answer is positive. Namely, consider the system of linear equations given by (2.13) where e_{ijk}, $f_{ij\ell}$, $g_{ik\ell}$ and $h_{jk\ell}$ are the unknowns. This system has m^2n^2 equations and $2m^2n+2mn^2$ unknowns. Then Q is linearizable if and only if our system has a solution. Furthermore, the linearization is given by

$$a_{ij}:=\sum_{k=1}^{n}(e_{ijk}+f_{ijk}) \qquad \text{and} \qquad b_{k\ell}:=\sum_{i=1}^{m}(g_{ik\ell}+h_{ik\ell}).$$

Corollary 8. *Let Q be a cost array of the BAP. Then deciding whether Q is linearizable and finding the linearization matrices A and B, can be done in polynomial time.*

2.2.2 Cost array of rank one

Recall that the cost array Q of BAP can be viewed as an $m^2\times n^2$ cost matrix. The rank of Q, when Q is viewed as a matrix, is at most r if and only if there exist some $m\times m$ matrices $A^p=(a^p_{ij})$ and $n\times n$ matrices $B^p=(b^p_{ij})$, $p=1,\ldots,r$, such that

$$q_{ijk\ell}=\sum_{p=1}^{r}a^p_{ij}b^p_{k\ell} \qquad (2.16)$$

for all $i, j \in M$, $k, \ell \in N$. We say that (2.16) is a *factored form* of Q. Then solving a BAP instance (Q, C, D), where matrix Q is of fixed rank r, is the problem of minimizing

$$f(\mathbf{x}, \mathbf{y}) = \sum_{p=1}^{r} \left(\sum_{i,j=1}^{m} a_{ij}^{p} x_{ij} \right) \left(\sum_{k,\ell=1}^{n} b_{k\ell}^{p} y_{k\ell} \right) + \sum_{i,j=1}^{m} c_{ij} x_{ij} + \sum_{k,\ell=1}^{n} d_{k\ell} y_{k\ell}, \tag{2.17}$$

such that $\mathbf{x} \in \mathcal{X}$, $\mathbf{y} \in \mathcal{Y}$.

Theorem 9. *If the $m^2 \times n^2$ matrix Q is of rank one and is given in factored form, and either C or D is a sum matrix, then the corresponding BAP instance (Q, C, D) is solvable in $O(m^3 + n^3)$ time.*

Proof. Let Q be of rank one, i.e. $q_{ijk\ell} = a_{ij} b_{k\ell}$ for all $i, j \in M$, $k, \ell \in N$. Furthermore, without loss of generality, assume that D is a sum matrix. Then there exist n-vectors $S = (s_i)$ and $T = (t_i)$ such that $d_{ij} = s_i + t_j$. Note that the value $\sum_{k,\ell=1}^{n} d_{k\ell} y_{k\ell}$ is the same for all $\mathbf{y} \in \mathcal{Y}$, hence solving the BAP instance (Q, C, D) is equivalent to solving the BAP instance (Q, C, O^n), that is, we need to minimize

$$f'(\mathbf{x}, \mathbf{y}) := \left(\sum_{i,j=1}^{m} a_{ij} x_{ij} \right) \left(\sum_{k,\ell=1}^{n} b_{k\ell} y_{k\ell} \right) + \sum_{i,j=1}^{m} c_{ij} x_{ij}, \tag{2.18}$$

subject to $\mathbf{x} \in \mathcal{X}$ and $\mathbf{y} \in \mathcal{Y}$.

Let $\mathbf{x}^* \in \mathcal{X}$ and $\mathbf{y}^* \in \mathcal{Y}$ be a solution that minimizes (2.18). Note that if $\sum_{i,j=1}^{m} a_{ij} x_{ij}^* > 0$, then $\mathbf{y}^*$ is an assignment that minimizes $\sum_{k,\ell=1}^{n} b_{k\ell} y_{k\ell}$. Analogously, if $\sum_{i,j=1}^{m} a_{ij} x_{ij}^* < 0$, then $\mathbf{y}^*$ is an assignment that maximizes $\sum_{k,\ell=1}^{n} b_{k\ell} y_{k\ell}$. If $\sum_{i,j=1}^{m} a_{ij} x_{ij}^* = 0$, then $\mathbf{y}^*$ can be arbitrary as it does not contributes to the objective value $f'(\mathbf{x}^*, \mathbf{y}^*)$. Hence, we only need to consider assignments $\mathbf{y}_{\min}, \mathbf{y}_{\max} \in \mathcal{Y}$ that minimize and maximize $\sum_{k,\ell=1}^{n} b_{k\ell} y_{k\ell}$, respectively. They are found in $O(n^3)$ time. Once $\mathbf{y}$ is fixed to $\bar{\mathbf{y}} \in \mathcal{Y}$, minimizing $f'(\mathbf{x}, \bar{\mathbf{y}})$ reduces to solving the linear assignment problem

$$f'(\mathbf{x}, \bar{\mathbf{y}}) = \sum_{i,j=1}^{m} \left(\bar{B} \cdot a_{ij} + c_{ij} \right) x_{ij},$$

where $\bar{B} := \sum_{k,\ell=1}^{n} b_{k\ell} \bar{y}_{k\ell}$. This can be done in $O(m^3)$ time, and we do it for $\bar{\mathbf{y}} = \mathbf{y}_{\min}$ and $\bar{\mathbf{y}} = \mathbf{y}_{\max}$. Better of the two will give us an optimal solution. □ □

2.3 Approximations

In the previous section we identified some classes of BAP for which an optimal solution can be found in polynomial time. In this section we develop and analyze various heuristic algorithms for BAP. In our analysis, we use approximation ratio [116] and domination ratio [53] to measure the quality of a heuristic solution.

2.3.1 A discretization procedure

Let $(\bar{\mathbf{x}}, \bar{\mathbf{y}})$ be a feasible solution to the bilinear program (BALP) obtained by relaxing constraints (2.6) of BAP to $0 \le x_{ij} \le 1$ for all i, j and $0 \le y_{k\ell} \le 1$ for all k, ℓ. As indicated earlier, BALP is equivalent to BAP in the sense that there exists an optimal solution $(\mathbf{x}, \mathbf{y})$ to BALP where $\mathbf{x} \in \mathcal{X}$ and $\mathbf{y} \in \mathcal{Y}$. But fractional solutions could also be optimal for BALP (although there always exists an optimal 0-1 solution). We present a simple discretization procedure such that from any solution $(\bar{\mathbf{x}}, \bar{\mathbf{y}})$ of BALP we can obtain a solution $(\mathbf{x}^*, \mathbf{y}^*)$ of BAP such that $f(\mathbf{x}^*, \mathbf{y}^*) \le f(\bar{\mathbf{x}}, \bar{\mathbf{y}})$. (Note that the notation $f(\mathbf{x}, \mathbf{y})$ was introduced for 0-1 vectors but naturally extends to any vectors.) This algorithm is useful in developing our FPTAS.

Given an instance (Q, C, D) and a BALP solution $(\bar{\mathbf{x}}, \bar{\mathbf{y}})$, find an optimal solution $\mathbf{x}^*$ of the LAP defined by the $m \times m$ cost matrix $H = (h_{ij})$ where

$$h_{ij} = c_{ij} + \sum_{k=1}^{n} \sum_{\ell=1}^{n} q_{ijk\ell} \bar{y}_{k\ell}. \tag{2.19}$$

Then choose $\mathbf{y}^*$ to be an optimal solution of the LAP defined by the $n \times n$ cost matrix $G = (g_{k\ell})$ where

$$g_{k\ell} = d_{k\ell} + \sum_{i=1}^{m} \sum_{j=1}^{m} q_{ijk\ell} x^*_{ij}. \tag{2.20}$$

The above discretization procedure is called *discretize-x optimize-y* (DxOy), as $\mathbf{y}^*$ is an optimal 0-1 assignment matrix when $\mathbf{x}$ is fixed at $\mathbf{x}^*$, and $\mathbf{x}^*$ is obtained by "discretizing" $\bar{\mathbf{x}}$. Naturally, *discretize-y optimize-x* (DyOx) procedure can be defined by swapping the order of operations on $\bar{\mathbf{x}}$ and $\bar{\mathbf{y}}$ (we omit the details). Since LAP can be solved in cubic running time, we have that the complexity of DxOy and DyOx procedures is $O(m^2n^2 + m^3 + n^3)$.

Similar procedures for the unconstrained bipartite boolean quadratic program and bipartite quadratic assignment problem were investigated in [108] and [37], respectively.

Theorem 10. *If a feasible solution $(\mathbf{x}^*, \mathbf{y}^*)$ of BAP is obtained by DxOy or DyOx procedure from a feasible solution $(\bar{\mathbf{x}}, \bar{\mathbf{y}})$ of BALP, then $f(\mathbf{x}^*, \mathbf{y}^*) \le f(\bar{\mathbf{x}}, \bar{\mathbf{y}})$.*

Proof. Let $(\bar{\mathbf{x}}, \bar{\mathbf{y}})$ be a feasible solution of BALP. Then

$$\begin{aligned} f(\bar{\mathbf{x}}, \bar{\mathbf{y}}) &= \sum_{i\in M} \sum_{j\in M} \sum_{k\in N} \sum_{\ell\in N} q_{ijk\ell} \bar{x}_{ij} \bar{y}_{k\ell} + \sum_{i\in M} \sum_{j\in M} c_{ij} \bar{x}_{ij} + \sum_{k\in N} \sum_{\ell\in N} d_{k\ell} \bar{y}_{k\ell} \\ &= \sum_{i\in M} \sum_{j\in M} \left(\sum_{k\in N} \sum_{\ell\in N} q_{ijk\ell} \bar{y}_{k\ell} + c_{ij} \right) \bar{x}_{ij} + \sum_{k\in N} \sum_{\ell\in N} d_{k\ell} \bar{y}_{k\ell} \\ &= \sum_{i\in M} \sum_{j\in M} h_{ij} \bar{x}_{ij} + \sum_{k\in N} \sum_{\ell\in N} d_{k\ell} \bar{y}_{k\ell}, \end{aligned}$$

where h_{ij}'s are defined as in (2.19). Since the constraint matrix of LAP is totally unimodular, an optimal integral solution is no worse than any non-integral solution, and therefore $\sum_{i\in M}\sum_{j\in M} h_{ij}\bar{x}_{ij} \geq \sum_{i\in M}\sum_{j\in M} h_{ij}x^*_{ij}$. Hence

$$
\begin{aligned}
f(\bar{\mathbf{x}},\bar{\mathbf{y}}) &\geq \sum_{i\in M}\sum_{j\in M}\left(\sum_{k\in N}\sum_{\ell\in N} q_{ijk\ell}\bar{y}_{k\ell} + c_{ij}\right)x^*_{ij} + \sum_{k\in N}\sum_{\ell\in N} d_{k\ell}\bar{y}_{k\ell} \\
&= \sum_{i\in M}\sum_{j\in M} c_{ij}x^*_{ij} + \sum_{k\in N}\sum_{\ell\in N}\left(\sum_{i\in M}\sum_{j\in M} q_{ijk\ell}x^*_{ij} + d_{k\ell}\right)\bar{y}_{k\ell} \\
&= \sum_{i\in M}\sum_{j\in M} c_{ij}x^*_{ij} + \sum_{k\in N}\sum_{\ell\in N} g_{k\ell}\bar{y}_{k\ell}
\end{aligned}
$$

where g_{ij}'s are defined as in (2.20). Again, we have that $\sum_{k\in N}\sum_{\ell\in N} g_{k\ell}\bar{y}_{k\ell} \geq \sum_{k\in N}\sum_{\ell\in N} g_{k\ell}y^*_{k\ell}$, hence

$$
\begin{aligned}
f(\bar{\mathbf{x}},\bar{\mathbf{y}}) &\geq \sum_{i\in M}\sum_{j\in M} c_{ij}x^*_{ij} + \sum_{k\in N}\sum_{\ell\in N}\left(\sum_{i\in M}\sum_{j\in M} q_{ijk\ell}x^*_{ij} + d_{k\ell}\right)y^*_{k\ell} \\
&= \sum_{i\in M}\sum_{j\in M}\sum_{k\in N}\sum_{\ell\in N} q_{ijk\ell}x^*_{ij}y^*_{k\ell} + \sum_{i\in M}\sum_{j\in M} c_{ij}x^*_{ij} + \sum_{k\in N}\sum_{\ell\in N} d_{k\ell}y^*_{k\ell} \\
&= f(\mathbf{x}^*,\mathbf{y}^*).
\end{aligned}
$$

The proof for DyOx works in the same way. □ □

2.3.2 FPTAS for BAP with fixed rank of Q

Recall that BAP does not admit a polynomial time α-approximation algorithm, unless P=NP. However, we now observe that in the case when the cost matrix Q is of fixed rank, there exists an FPTAS. This follows from the results of Mittal and Schulz in [99], where the authors present an FPTAS for a class of related optimization problems.

Theorem 11 (Mittal, Schulz [99])**.** *Let k be a fixed positive integer, and let $\mathcal{OP}^k$ be an optimization problem*

$$
\begin{aligned}
\min \quad & g(\mathbf{z}) = h(E_1^T\mathbf{z}, E_2^T\mathbf{z}, \ldots, E_k^T\mathbf{z}) \\
s.t. \quad & \mathbf{z} \in \mathcal{P},
\end{aligned}
$$

where $\mathcal{P} \subseteq \mathbb{R}^n$ is a polytope, and function $h\colon \mathbb{R}^k_+ \to \mathbb{R}$ and vectors $E_i \in \mathbb{R}^n$, $i = 1,\ldots,k$, are such that the conditions

1. $h(\alpha) \leq h(\beta) \qquad \forall\alpha,\beta \in \mathbb{R}^k_+,\ s.t.\ \alpha_i \leq \beta_i$ *for all* $i = 1,2,\ldots,k$,
2. $h(\lambda\alpha) \leq \lambda^c h(\alpha) \quad \forall\alpha \in \mathbb{R}^k_+, \lambda > 1$ *and some constant* c,
3. $E_i^T\mathbf{z} > 0 \qquad$ *for* $i = 1,2,\ldots,k$ *and* $\mathbf{z} \in \mathcal{P}$,

are satisfied. Then there exists an FPTAS for $\mathcal{OP}^k$.

For the description of the FPTAS see [99].

Corollary 12. *Let r be a fixed integer. Then there exists an FPTAS for the BAP on a cost matrix Q of rank r (see (2.17)), if $\sum_{i,j=1}^{m} a_{ij}^p x_{ij} > 0$, $\sum_{k,\ell=1}^{n} b_{k\ell}^p y_{k\ell} > 0$, $p = 1, 2, \ldots, r$, and $\sum_{i,j=1}^{m} c_{ij} x_{ij} > 0$, $\sum_{k,\ell=1}^{n} d_{k\ell} y_{e\ell} > 0$ are satisfied for all $\mathbf{x} \in \mathcal{X}$, $\mathbf{y} \in \mathcal{Y}$.*

Proof. The relaxations of the problems described in the statement of the corollary fall into the class of problems $\mathcal{OP}^k$ described in Theorem 11. Namely, given such instance (Q, C, D) of BALP, we can express it as an $\mathcal{OP}^k$ where $k := 2r + 1$, $h(E_1^T \mathbf{z}, E_2^T \mathbf{z}, \ldots, E_{2r+1}^T \mathbf{z}) := \sum_{i=1}^{r} (E_{2i-1}^T \mathbf{z})(E_{2i}^T \mathbf{z}) + E_{2r+1}^T \mathbf{z}$, polytop $\mathcal{P} \subset \mathbb{R}^{m^2+n^2}$ is the convex hull of $\{\mathbf{x} \otimes \mathbf{y} : \mathbf{x} \in \mathcal{X}, \mathbf{y} \in \mathcal{Y}\}$, and

$$E_i := \begin{cases} A^{(i+1)/2} \otimes O^n & \text{if } i \text{ is odd,} \\ O^m \otimes B^{i/2} & \text{if } i \text{ is even,} \\ C \otimes D & \text{if } i = 2r + 1, \end{cases}$$

where $A^p = (a_{ij}^p)$, $B^p = (b_{ij}^p)$, O^ℓ denotes the $\ell \times \ell$ null-matrix, and $\otimes$ denotes an operation of concatenating $m \times m$ and $n \times n$ matrices into $(m^2 + n^2)$-vectors. Furthermore, condition (iii) is mandated in the corollary, which then implies (i). Condition (ii) can be checked easily. Hence, according to Theorem 11, there exist an FPTAS that approximates our BALP with fixed rank matrix Q. Theorem 10 implies that by discretizing fractional FPTAS solutions using DxOy (or DyOx), we get an FPTAS for the corresponding BAP with fixed rank Q. □ □

2.3.3 Domination analysis

The negative result of Theorem 1 precludes potential success in developing approximation algorithms for BAP with constant approximation ratio, unless additional strong assumptions are made. We now show that domination ratio [53] can be used to establish some performance guarantee for BAP heuristics.

Domination analysis has been successfully pursued by many researchers to provide performance guarantee of heuristics for various combinatorial optimization problems [4, 6, 36, 53, 61, 63, 65, 67, 66, 71, 89, 84, 105, 106, 108, 115, 118, 119, 117, 130, 131, 133]. Domination analysis is also linked to exponential neighborhoods [39] and very large-scale neighborhood search [3, 98]. Domination analysis results similar to what is presented here have been obtained for the *bipartite quadratic assignment problems* [37] and the *bipartite boolean quadratic programs* [108].

Given an instance (Q, C, D) of a BAP, let $\mathcal{A}(Q, C, D)$ be the average of the objective function values of all feasible solutions.

Theorem 13. $\mathcal{A}(Q, C, D) = \dfrac{1}{mn}\sum_{i=1}^{m}\sum_{j=1}^{m}\sum_{k=1}^{n}\sum_{\ell=1}^{n} q_{ijk\ell} + \dfrac{1}{m}\sum_{i=1}^{m}\sum_{j=1}^{m} c_{ij} + \dfrac{1}{n}\sum_{k=1}^{n}\sum_{\ell=1}^{n} d_{k\ell}.$

The proof of Theorem 13 follows from the observation that a cost $q_{ijk\ell}$ appears in the objective value sum of $(\mathbf{x}, \mathbf{y})$ if and only if $x_{ij} = 1$ and $y_{k\ell} = 1$, and there is exactly $(m-1)!(n-1)!$ such solutions $(\mathbf{x}, \mathbf{y})$ in $\mathcal{F}$. We omit details. Similar formulas for the QAP have been investigated by Graves and Whinstone [59].

Consider the fractional solution $(\mathbf{x}, \mathbf{y})$ where $x_{ij} = 1/m$ for all $i, j \in M$, and let $y_{ij} = 1/n$ for all $i, j \in N$. Then $(\mathbf{x}, \mathbf{y})$ is a feasible solution to BALP. It is also easy to see that $f(\mathbf{x}, \mathbf{y}) = \mathcal{A}(Q, C, D)$. From Theorem 10 it follows that a solution $(\mathbf{x}^*, \mathbf{y}^*)$ of BAP constructed from $(\mathbf{x}, \mathbf{y})$ by applying DxOy or DyOx, satisfies $f(\mathbf{x}^*, \mathbf{y}^*) \leq \mathcal{A}(Q, C, D)$. Therefore, a BAP feasible solution $(\mathbf{x}^*, \mathbf{y}^*)$ that satisfies $f(\mathbf{x}^*, \mathbf{y}^*) \leq \mathcal{A}(Q, C, D)$ can be obtained in $O(m^2n^2 + m^3 + n^3)$ time using DxOy or DyOx procedure.

A feasible solution $(\mathbf{x}, \mathbf{y})$ of BAP is said to be no better than average if $f(\mathbf{x}, \mathbf{y}) \geq A(Q, C, D)$. We will provide a lower bound for the number of feasible solutions that are no better than the average. Establishing corresponding results for QAP is an open problem [6, 36, 67, 117]. Given an instance (Q, C, D) of BAP, define

$$\mathcal{G}(Q, C, D) = \{(\mathbf{x}, \mathbf{y}) \in \mathcal{F} \,:\, f(\mathbf{x}, \mathbf{y}) \geq A(Q, C, D)\}.$$

Theorem 14. $|\mathcal{G}(Q, C, D)| \geq (m-1)!(n-1)!$.

Proof. Consider an equivalence relation $\sim$ on $\mathcal{F}$, where $(\mathbf{x}, \mathbf{y}) \sim (\mathbf{x}', \mathbf{y}')$ if and only if there exist $a \in \{0, 1, \ldots, m-1\}$ and $b \in \{0, 1, \ldots, n-1\}$ such that $x_{ij} = x'_{i(j+a \mod m)}$ for all i, j, and $y_{k\ell} = y'_{k(\ell+b \mod n)}$ for all k, ℓ. Note that $\sim$ partitions $\mathcal{F}$ into $(m-1)!(n-1)!$ equivalence classes, each of size mn. Fix S to be one such equivalence class. For every $i, j \in M$ and $k, \ell \in N$, there are exactly n solutions in S for which $x_{ij} = 1$, and there are exactly m solutions in S for which $y_{k\ell} = 1$. Furthermore, for every $i, j \in M$ and $k, \ell \in N$, there is exactly one solution in S for which $x_{ij}y_{k\ell} = 1$. Therefore

$$\sum_{(\mathbf{x},\mathbf{y})\in S} f(\mathbf{x}, \mathbf{y}) = \sum_{\substack{(i,j,k,\ell)\in \\ M\times M\times N\times N}} q_{ijk\ell} + n\sum_{\substack{(i,j)\in \\ M\times M}} c_{ij} + m\sum_{\substack{(k,\ell)\in \\ N\times N}} d_{k\ell} = mn\mathcal{A}(Q, C, D).$$

Because $|S| = mn$, there must be at least one $(\mathbf{x}, \mathbf{y}) \in S$ such that $f(\mathbf{x}, \mathbf{y}) \geq \mathcal{A}(Q, C, D)$. There is one such solution for each of the $(m-1)!(n-1)!$ equivalent classes and this concludes the proof. □ □

An algorithm that is guaranteed to return a solution with the objective function value at most $\mathcal{A}(Q, C, D)$ guarantees a solution that is no worse than $(m-1)!(n-1)!$ solutions. Thus, the domination ratio [53, 37] of such an algorithm is $\frac{1}{mn}$.

Here and later in the write-up we use the notation of $x_{i(j+a \mod m)}$ in a sense that, if $(j+a) \mod m = 0$, we then assume it to refer to the variable x_{im}. Similar assumptions will be made for the other index of x_{ij} and variables y_{kl} to improve the clarity of presentation.

The bound presented in Theorem 14 is tight. To see this, let cost arrays Q, C, D be such that all of their elements are 0, except $q_{i'j'k'\ell'} = 1$ for some fixed i', j', k', ℓ'. The tightness follows from the fact that exactly one element from every equivalence class defined by $\sim$ is no better than average.

The proof of Theorem 14 also provides us a way to construct an $(\mathbf{x}, \mathbf{y}) \in \mathcal{F}$ such that $f(\mathbf{x}, \mathbf{y}) \leq \mathcal{A}(Q, C, D)$. We showed that in every equivalence class defined by $\sim$ there is a feasible solution with the objective function value greater than or equal to $\mathcal{A}(Q, C, D)$. By the same reasoning it follows that in every such class there is a feasible solution with objective function value less than or equal to $\mathcal{A}(Q, C, D)$. For example, given $a \in M$, $b \in N$ let $(\mathbf{x}^a, \mathbf{y}^b) \in \mathcal{F}$ be defined as

$$x^a_{ij} = \begin{cases} 1 & \text{if } j = i + a \mod m, \\ 0 & \text{otherwise} \end{cases} \quad \text{and} \quad y^b_{k\ell} = \begin{cases} 1 & \text{if } \ell = k + b \mod n, \\ 0 & \text{otherwise.} \end{cases}$$

Then $(\mathbf{x}^{a_1}, \mathbf{y}^{b_1}) \sim (\mathbf{x}^{a_2}, \mathbf{y}^{b_2})$ for every $a_1, a_2 \in M$ and $b_1, b_2 \in N$, and

$$f(\mathbf{x}^a, \mathbf{y}^b) = \sum_{i \in M, k \in N} q_{i(i+a \mod m)k(k+b \mod n)} + \sum_{i \in M} c_{i(i+a \mod m)} + \sum_{k \in N} d_{k(k+b \mod n)}.$$

Corollary 15. *For any instance (Q, C, D) of BAP*

$$\min_{a \in M, b \in N} \{f(\mathbf{x}^a, \mathbf{y}^b)\} \leq \mathcal{A}(Q, C, D) \leq \max_{a \in M, b \in N} \{f(\mathbf{x}^a, \mathbf{y}^b)\}.$$

Note that any equivalence class defined by $\sim$ can be used to obtain the type of inequalities above. Corollary 15 provides another way to find a feasible solution to BAP with objective function value no worse than $\mathcal{A}(Q, C, D)$ in $O(m^2 n^2)$ time. This is a running time improvement when compared to the approach using DxOy or DyOx described above.

Interestingly, unlike the average, computing the median of the objective function values of feasible solutions of BAP is NP-hard.

Theorem 16. *Finding a median of the objective function values for the BAP is NP-hard.*

Proof. We will describe a reduction from the NP-complete PARTITION problem: Given n positive integers $a_1, a_2, \ldots, a_n$, determine if there exists a partition of $N = \{1, 2, \ldots, n\}$ into S_1 and S_2 such that $\sum_{i \in S_1} a_i = \sum_{i \in S_2} a_i$. From a PARTITION instance I, we will create a BAP instance (Q, C, D), such that a median of objective values of BAP is $\sum_{i \in N} a_i/2$ if and only if the starting PARTITION instance I is a 'yes' instance.

Without loss of generality we assume than n is even. Let the BAP instance (Q, C, D) be such that C is the 2×2 null matrix, D is the $n \times n$ null matrix, and the $2 \times 2 \times n \times n$ array Q is given by

$$q_{ijk\ell} = \begin{cases} \frac{a_k}{2} & \text{if } i = j \text{ and } k = \ell, \\ \frac{a_k}{2} & \text{if } i \neq j \text{ and } k \neq \ell, \\ 0 & \text{otherwise.} \end{cases}$$

We can partition the set of feasible solutions $\mathcal{F}$ into pairs of solutions $(\mathbf{x}, \mathbf{y})$, $(\mathbf{x}', \mathbf{y}')$ by the rule $x_{ij} = x'_{i(j+1 \mod 2)}$ and $y_{k\ell} = y'_{k\ell}$, for all $i, j \in \{1, 2\}$ and $k, \ell \in N$. For every such pair of solutions, either $f(\mathbf{x}, \mathbf{y}) \leq \sum_{i=1}^{n} a_i/2 \leq f(\mathbf{x}', \mathbf{y}')$ or $f(\mathbf{x}', \mathbf{y}') \leq \sum_{i=1}^{n} a_i/2 \leq f(\mathbf{x}, \mathbf{y})$. Hence, if there is a solution $(\mathbf{x}, \mathbf{y})$ for which $f(\mathbf{x}, \mathbf{y}) = \sum_{i=1}^{n} a_i/2$, then $\sum_{i=1}^{n} a_i/2$ is a median objective value. Value K appears as an objective value of some solution for (Q, C, D) if and only if there exists $S \subseteq \{1, 2, \ldots, n\}$ such that $\sum_{i \in S} a_i = K$. Therefore, the theorem follows from NP-completeness of the PARTITION problem. □

For a given instance I of an optimization problem, let $\mathbf{x}^\Gamma \in \mathcal{F}(I)$ be a solution produced by an algorithm Γ, where $\mathcal{F}(I)$ denotes the set of feasible solutions for the instance I. Let $\mathcal{G}^\Gamma(I) = \{\mathbf{x} \in \mathcal{F}(I) : f(\mathbf{x}) \geq f(\mathbf{x}^\Gamma)\}$ where $f(\mathbf{x})$ denotes the objective function value of $\mathbf{x}$. Furthermore, let $\mathcal{I}$ be the collection of all instances of the problem with a fixed instance size. Then

$$\inf_{I \in \mathcal{I}} \left| \mathcal{G}^\Gamma(I) \right| \quad \text{and} \quad \inf_{I \in \mathcal{I}} \frac{|\mathcal{G}^\Gamma(I)|}{|\mathcal{F}(I)|},$$

are called *domination number* and *domination ratio* of Γ, respectively [4, 53].

As we discussed above, procedure DxOy (and DyOx) and Corollary 15 give us two BAP heuristics for which the domination number $(m-1)!(n-1)!$ and the domination ratio $\frac{1}{mn}$ is guarantied by Theorem 14. Lastly, we give an upper bound on the domination ratio for any polynomial time heuristic algorithm for the BAP. The result can be shown following the main idea in [108, 37].

Theorem 17. *For any fixed rational number $\beta > 1$ no polynomial time algorithm for BAP can have domination number greater than $m!n! - \lceil \frac{m}{\beta} \rceil ! \lceil \frac{n}{\beta} \rceil !$, unless P=NP.*

Proof. Let β be a rational number such that $\beta > 1$. We show that a polynomial algorithm Γ for BAP, with domination number at least $m!n! - \lceil \frac{m}{\beta} \rceil ! \lceil \frac{n}{\beta} \rceil ! + 1$, can be used to compute an optimal solution of BAP. Consider an arbitrary BAP instance (Q, C, D). Let a and b be two natural numbers such that $\beta = \frac{a}{b} > 1$, and let $Q^* = (q^*_{ijk\ell})$ be an $abm \times abm \times abn \times abn$ array such that

$$q^*_{ijk\ell} = \begin{cases} q_{ijk\ell} & \text{if } i, j \in M \text{ and } k, \ell \in N, \\ 0 & \text{otherwise.} \end{cases}$$

Furthermore, let L be a large number and let $C^* = (c^*_{ij})$ and $D^* = (d^*_{k\ell})$ be $abm \times abm$ and $abn \times abn$ matrices respectively, such that

$$c^*_{ij} = \begin{cases} c_{ij} & \text{if } i,j \in M, \\ 0 & \text{if } i,j \notin M, \\ L & \text{otherwise} \end{cases} \quad \text{and} \quad d^*_{k\ell} = \begin{cases} d_{k\ell} & \text{if } k,\ell \in N, \\ 0 & \text{if } k,\ell \notin N, \\ L & \text{otherwise.} \end{cases}$$

Note that BAP instances (Q^*, C^*, D^*) and (Q, C, D) are equivalent. In particular, from any optimal solution for (Q^*, C^*, D^*) an optimal solution for (Q, C, D) can be recovered. Moreover, the number of optimal solutions for (Q^*, C^*, D^*) is at least $(abm - m)!(abn - n)!$. Therefore, the number of non-optimal solutions is at most $(abm)!(abn)! - (abm - m)!(abn - n)!$. Let $(\mathbf{x}', \mathbf{y}')$ denote the output of Γ, and assume that $(\mathbf{x}', \mathbf{y}')$ is no worse than $(abm)!(abn)! - \lceil \frac{abm}{\beta} \rceil ! \lceil \frac{abn}{\beta} \rceil ! + 1 = (abm)!(abn)! - (b^2 m)!(b^2 n)! + 1$ solutions. From $a > b$, it follows that $(abm - m)!(abn - n)!$ is greater than or equal to $(b^2 m)!(b^2 n)!$, therefore $(\mathbf{x}', \mathbf{y}')$ is an optimal solution. Using $(\mathbf{x}', \mathbf{y}')$ we can now find an optimal solution for (Q, C, D), which contradict the fact that BAP is NP-hard. □

2.4 Integer programming linearizations

We now present several linearization reformulations of BAP, where the problem will appear as a mixed-integer linear program. This enables the usage of MILP solvers to tackle the problem.

Similar to QAP linearization in [94], we introduce $m^2 n^2$ binary variables $z_{ijk\ell}$, such that

$$z_{ijk\ell} = x_{ij} y_{k\ell}, \qquad i,j = 1, \dots, m, \quad k,\ell = 1, \dots, n. \tag{2.21}$$

Now an integer linear program equivalent to BAP (2.1) - (2.6) is:

$$\begin{aligned} \text{Minimize} \quad & \sum_{i=1}^{m} \sum_{j=1}^{m} \sum_{k=1}^{n} \sum_{\ell=1}^{n} q_{ijk\ell} z_{ijk\ell} + \sum_{i=1}^{m} \sum_{j=1}^{m} c_{ij} x_{ij} + \sum_{k=1}^{n} \sum_{\ell=1}^{n} d_{k\ell} y_{k\ell} & (2.22) \\ \text{subject to} \quad & (2.2) - (2.5), \\ & \sum_{i=1}^{m} \sum_{j=1}^{m} \sum_{k=1}^{n} \sum_{\ell=1}^{n} z_{ijk\ell} = mn, & (2.23) \\ & z_{ijk\ell} \leq \frac{x_{ij} + y_{k\ell}}{2} \qquad i,j = 1, \dots, m, \quad k,\ell = 1, \dots, n, & (2.24) \\ & x_{ij},\ y_{k\ell},\ z_{ijk\ell} \in \{0,1\} \qquad i,j = 1, \dots, m, \quad k,\ell = 1, \dots, n. & (2.25) \end{aligned}$$

Alternatively, we can substitute (2.23) by $m^2 n^2$ constraints

$$z_{ijk\ell} \geq x_{ij} + y_{k\ell} - 1 \qquad i,j = 1,\dots,m, \quad k,\ell = 1,\dots,n. \tag{2.26}$$

This kind of linearization [49] also introduces m^2n^2 variables

$$z_{ijk\ell} = x_{ij}y_{k\ell}, \qquad i,j = 1,\dots,m, \quad k,\ell = 1,\dots,n. \tag{2.27}$$

The BAP is rewritten then as the following integer linear program:

$$\begin{aligned}
\text{Minimize} \quad & \sum_{i=1}^{m}\sum_{j=1}^{m}\sum_{k=1}^{n}\sum_{\ell=1}^{n} q_{ijk\ell}z_{ijk\ell} + \sum_{i=1}^{m}\sum_{j=1}^{m} c_{ij}x_{ij} + \sum_{k=1}^{n}\sum_{\ell=1}^{n} d_{k\ell}y_{k\ell} && (2.28)\\
\text{subject to} \quad & (2.2) - (2.5), && \\
& \sum_{j=1}^{m} z_{ijk\ell} = y_{k\ell} \qquad i = 1,2,\dots,m, \quad k,\ell = 1,\dots,n, && (2.29)\\
& \sum_{i=1}^{m} z_{ijk\ell} = y_{k\ell} \qquad j = 1,2,\dots,m, \quad k,\ell = 1,\dots,n, && (2.30)\\
& \sum_{\ell=1}^{n} z_{ijk\ell} = x_{ij} \qquad k = 1,2,\dots,n, \quad i,j = 1,\dots,m, && (2.31)\\
& \sum_{k=1}^{n} z_{ijk\ell} = x_{ij} \qquad \ell = 1,2,\dots,n, \quad i,j = 1,\dots,m, && (2.32)\\
& x_{ij},\ y_{k\ell} \in \{0,1\} \qquad i,j = 1,\dots,m, \quad k,\ell = 1,\dots,n. && (2.33)\\
& z_{ijk\ell} \in \{0,1\} \qquad i,j = 1,\dots,m, \quad k,\ell = 1,\dots,n. && (2.34)
\end{aligned}$$

Notice that relaxing (2.34) to $0 \leq z_{ijk\ell} \leq 1$ maintains equivalence to BAP.

Extending the ideas of the previous linearization we multiply each of (2.29) - (2.32) by respectively x_{pq} and y_{rs} to get

$$\sum_{j=1}^{m} x_{pq}z_{ijk\ell} = z_{pqk\ell} \qquad i,p,q = 1,2,\dots,m, \quad k,\ell = 1,\dots,n, \tag{2.35}$$

$$\sum_{i=1}^{m} x_{pq}z_{ijk\ell} = z_{pqk\ell} \qquad j,p,q = 1,2,\dots,m, \quad k,\ell = 1,\dots,n, \tag{2.36}$$

$$\sum_{\ell=1}^{n} z_{ijk\ell}y_{rs} = z_{ijrs} \qquad k,r,s = 1,2,\dots,n, \quad i,j = 1,\dots,m, \tag{2.37}$$

$$\sum_{k=1}^{n} z_{ijk\ell}y_{rs} = z_{ijrs} \qquad \ell,r,s = 1,2,\dots,n, \quad i,j = 1,\dots,m. \tag{2.38}$$

Now we linearize by introducing $m^4n^2 + m^2n^4$ additional real variables:

$$v_{pqijk\ell} = x_{pq} z_{ijk\ell}, \qquad p,q,i,j = 1,\ldots,m, \quad k,\ell = 1,\ldots,n, \tag{2.39}$$

$$w_{ijk\ell rs} = z_{ijk\ell} y_{rs}, \qquad i,j = 1,\ldots,m, \quad k,\ell,r,s = 1,\ldots,n. \tag{2.40}$$

We now get the following MILP equivalent to our original BAP:

$$\begin{aligned}
\text{Minimize} \quad & (2.28) \\
\text{subject to} \quad & (2.2)-(2.5), (2.29)-(2.32), \\
& \sum_{j=1}^{m} v_{pqijk\ell} = z_{pqk\ell} \qquad i,p,q = 1,2,\ldots,m, \quad k,\ell = 1,\ldots,n, && (2.41) \\
& \sum_{i=1}^{m} v_{pqijk\ell} = z_{pqk\ell} \qquad j,p,q = 1,2,\ldots,m, \quad k,\ell = 1,\ldots,n, && (2.42) \\
& \sum_{\ell=1}^{n} w_{ijk\ell rs} = z_{ijrs} \qquad k,r,s = 1,2,\ldots,n, \quad i,j = 1,\ldots,m, && (2.43) \\
& \sum_{k=1}^{n} w_{ijk\ell rs} = z_{ijrs} \qquad \ell,r,s = 1,2,\ldots,n, \quad i,j = 1,\ldots,m, && (2.44) \\
& x_{ij},\ y_{k\ell} \in \{0,1\} \qquad i,j = 1,\ldots,m, \quad k,\ell = 1,\ldots,n. && (2.45) \\
& 0 \le z_{ijk\ell}, v_{pqijk\ell}, w_{ijk\ell rs} \le 1 \qquad p,q,i,j = 1,\ldots,m, \quad k,\ell,r,s = 1,\ldots,n. && (2.46)
\end{aligned}$$

Going for linearizations of higher levels will allow for a stronger bounds from the linear programming relaxation, however, extra computational costs will be induced.

Quadratic term from the objective function (2.1) can be rewritten as

$$\sum_{i=1}^{m}\sum_{j=1}^{m}\sum_{k=1}^{n}\sum_{\ell=1}^{n} q_{ijk\ell} x_{ij} y_{k\ell} = \sum_{i=1}^{m}\sum_{j=1}^{m} x_{ij} \sum_{k=1}^{n}\sum_{\ell=1}^{n} q_{ijk\ell} y_{k\ell}. \tag{2.47}$$

Now, following the QAP linearization from [79], we define m^2 real variables:

$$z_{ij} = x_{ij} \sum_{k=1}^{n}\sum_{\ell=1}^{n} q_{ijk\ell} y_{k\ell} \qquad i,j = 1,\ldots,m, \tag{2.48}$$

Moreover, let $a_{ij} = \sum_{k=1}^{n}\sum_{\ell=1}^{n} q_{ijk\ell}$ for $i,j = 1,\ldots,m$. Note that coefficients $q_{ijk\ell}$ could be considered positive without loosing generality. The mixed-integer linear program equivalent to BAP can be written as:

$$
\begin{aligned}
\text{Minimize} \quad & \sum_{i=1}^{m}\sum_{j=1}^{m} z_{ij} && (2.49)\\
\text{subject to} \quad & (2.2)-(2.5), \\
& z_{ij} + (1-x_{ij})a_{ij} \geq \sum_{k=1}^{n}\sum_{\ell=1}^{n} q_{ijk\ell}y_{k\ell} \qquad i,j = 1,2,\ldots,m, && (2.50)\\
& x_{ij},\ y_{k\ell} \in \{0,1\} \qquad i,j = 1,\ldots,m, \quad k,\ell = 1,\ldots,n. && (2.51)\\
& z_{ijk\ell} \geq 0 \qquad i,j = 1,\ldots,m, \quad k,\ell = 1,\ldots,n. && (2.52)
\end{aligned}
$$

2.5 Conclusion

We presented a systematic study of complexity aspects of BAP defined on the data set (Q, C, D) and size parameters m and n. BAP generalizes the well known quadratic assignment problem, the three dimensional assignment problem, and the disjoint matching problem. We show that BAP is NP-hard if $m = O(\sqrt[r]{n})$, for some fixed r, but is solvable in polynomial time if $m = O(\frac{\log n}{\log\log n})$. Further, we establish that BAP cannot be approximated within a constant factor unless P=NP even if Q is diagonal; but when the rank of Q is fixed, BAP is observed to admit FPTAS. When the rank of Q is one and C or D is a sum matrix, BAP is shown to be solvable in polynomial time. In contrast, QAP with a diagonal cost matrix is just the linear assignment problem which is solvable in polynomial time. We also provide a characterization of BAP instances equivalent to two linear assignment problems. Same as in the case of QAP, linearizable instances yield a class of polynomially solvable special cases. Various results leading to performance guarantee of heuristics from the domination analysis point of view are presented. In particular, we showed that a feasible solution with objective function value no worse than that of $(m-1)!(n-1)!$ solutions can be identified efficiently, whereas computing a solution whose objective function value is no worse than that of $m!n! - \lceil \frac{m}{\beta} \rceil ! \lceil \frac{n}{\beta} \rceil !$ solutions is NP-hard for any fixed rational number $\beta > 1$. As a by-product, we have a closed form expression to compute the average of the objective function values of all solutions, but the median of the solution values cannot be identified in polynomial time, unless P=NP.

Chapter 3

Bilinear Assignment Problem: Theoretical and Experimental Analysis of Algorithms

In this chapter we discuss the results of our experimentally focused study on the model [121].

We present various neighborhoods associated with a feasible solution of BAP and analyze their theoretical properties in the context of local search algorithms, particularly on the worst case behavior. Some of these neighborhoods are of exponential size but can be searched for an improving solution in polynomial time. Local search algorithms with such *very large scale neighborhoods (VLSN)* proved to be an effective solution approach for many hard combinatorial optimization problems [2, 3]. We also present extensive experimental results by embedding these neighborhoods within a *variable neighborhood search (VNS)* framework in addition to the standard and multi-start VLSN local search. Some very fast construction heuristics are also provided along with experimental analysis. Although local search and variable neighborhood search are well known algorithmic paradigms that are thoroughly investigated in the context of various combinatorial optimization problems, to achieve effectiveness and obtain superior outcomes variable neighborhood search algorithms needs to exploit special problem structures that efficiently link the various neighborhoods under consideration. In this sense, developing variable neighborhood search algorithms is always intriguing, especially when it comes to new optimization problems having several well designed neighborhood structures with interesting properties. Our experimental analysis shows that the average behavior of the algorithms are much better and the established negative worst case performance hardly occurs. Such a conclusion can only be made by systematic experimentation, as we have done. On a balance of computational time and solution quality, a multi-start based VLSN local search became our proposed approach.

Although, by allowing significantly more time, a strategic variable neighborhood search outperformed this algorithm in terms of solution quality.

Several classes of neigborhood search structures are introduced for the problem along with some theoretical analysis. These neighborhoods are then explored within a local search and a variable neighborhood search frameworks with multistart to generate robust heuristic algorithms. Results of systematic experimental analysis have been presented which divulge the effectiveness of our algorithms. In addition, we present several very fast construction heuristics. Our experimental results disclosed some interesting properties of the BAP model, different from those of comparable models. This is the first thorough experimental analysis of algorithms on BAP. We have also introduced benchmark test instances that can be used for future experiments on exact and heuristic algorithms for the problem.

This chapter is organized as follows. In Section 3.1 we describe several construction heuristics for BAP that generate reasonable solutions, often quickly. In Section 3.2, we present various neighborhood structures and analyze their theoretical properties. We then (Section 3.3.1) describe in details specifics of our experimental setup as well as sets of instances that we have generated for the problem. The benchmark instances that we have developed are available upon request, for other researchers to further study the problem. The development of these test instances and best-known solutions is yet another contribution of this work. Sections 3.3.2 and 3.3.3 deal with experimental analysis of construction heuristics and local search algorithms. Our computational results disclose some interesting and unexpected outcomes, particularly when comparing standard local search with its multistart counterpart. In Section 3.3.4 we combine better performing construction heuristics and different local search algorithms to develop several variable neighborhood search algorithms and present comparison with our best performing multistart local search algorithm. Concluding remarks are presented in Section 3.4.

3.1 Construction heuristics

In this section, we consider heuristic algorithms that will generate solutions to BAP using various construction approaches. Such algorithms are useful in situations where solutions of reasonable quality are needed quickly. These algorithms can also be used to generate starting solutions for more complex improvement based algorithms.

Our first algorithm, called ***Random***, is the trivial approach of generating a feasible solution $(\mathbf{x}, \mathbf{y})$. Both $\mathbf{x}$ and $\mathbf{y}$ are selected as random assignments in uniform fashion. It should be noted that the expected value of the solution produced by *Random* is precisely $\mathcal{A}(Q, C, D)$.

Let us now discuss a different randomized technique, called ***RandomXYGreedy***. This algorithm builds a solution by randomly picking a 'not yet assigned' $i \in M$ or $k \in N$, and then setting x_{ij} or y_{kl} to 1 for a 'not yet assigned' $j \in M'$ or $l \in N'$ so that the total cost of the resulting *partial solution* is minimized. A pseudo-code of *RandomXYGreedy* is presented in Algorithm 1. Here and later in the paper we will present description of the algorithms by assuming that the input BAP instance (Q, C, D) has C and D as zero arrays. This restriction is for simplicity of presentation and does not affect neither the theoretical complexity of BAP nor the asymptotic computational complexity of the presented algorithms. It is easy to extend the algorithms to the general case in a straightforward way. The running time of *RandomXYGreedy* is $O(mn^2)$ as each addition to our solution is selected using quadratic number of computations. However, just reading the data for the Q matrix takes $O(m^2n^2)$ time. For the rest of the paper we will consider running time of our algorithms without including this input overhead.

Algorithm 1 *RandomXYGreedy*

```
Input: integers m, n; m × m × n × n array Q
Output: feasible solution to BAP
  x_ij ← 0 ∀i, j; y_kl ← 0 ∀k, l
  while not all i ∈ M and k ∈ N are assigned do
      randomly pick some i ∈ M or k ∈ N that is unassigned
      if i is picked then
          j' ← random j ∈ M that is unassigned; Δ' ← Σ_{k,l∈N} q_{ij'kl} y_kl
          for all j ∈ M that is unassigned do
              Δ ← Σ_{k,l∈N} q_{ijkl} y_kl              ▷ value change if i assigned to j
              if Δ < Δ' then
                  j' ← j; Δ' ← Δ
              end if
          end for
          x_{ij'} ← 1                                    ▷ assign i to j'
      else
          l' ← random l ∈ N that is unassigned; Δ' ← Σ_{i,j∈M} q_{ijkl'} x_ij
          for all l ∈ N that is unassigned do
              Δ ← Σ_{i,j∈M} q_{ijkl} x_ij              ▷ value change if k assigned to l
              if Δ < Δ' then
                  l' ← l; Δ' ← Δ
              end if
          end for
          y_{kl'} ← 1                                    ▷ assign k to l'
      end if
  end while
  return (x, y)
```

Our next algorithm is fully deterministic and is called ***Greedy*** (see Algorithm 2). This is similar to *RandomXYGreedy*, except that, at each iteration, we select the best available x_{ij} or y_{kl} to be added to the current partial solution. We start the algorithm by choosing the partial solution $x_{i'j'} = 1$ and $y_{k'l'} = 1$ where i', j', k', l' correspond to a smallest element in the array Q. The total running time of this heuristic is $O(n^3)$, considering that the position of the smallest $q_{i'j'k'l'}$ is provided.

Theorem 18. *The objective function value of a solution produced by the Greedy algorithm could be arbitrarily bad and could be worse than $\mathcal{A}(Q, C, D)$.*

Algorithm 2 *Greedy*

```
Input: integers m, n; m × m × n × n array Q
Output: feasible solution to BAP
  x_ij ← 0 ∀i, j; y_kl ← 0 ∀k, l
  i', j', k', l' ← arg min_{i,j∈M,k,l∈N} q_ijkl; x_i'j' ← 1; y_k'l' ← 1
  while not all i ∈ M and k ∈ N are assigned do
    Δ'_x ← ∞; Δ'_y ← ∞
    for all i ∈ M that is unassigned do
      for all j ∈ M that is unassigned do
        Δ ← Σ_{k,l∈N} q_ijkl y_kl              ▷ value change if i assigned to j
        if Δ < Δ'_x then
          i' ← i; j' ← j; Δ'_x ← Δ
        end if
      end for
    end for
    for all k ∈ N that is unassigned do
      for all l ∈ N that is unassigned do
        Δ ← Σ_{i,j∈M} q_ijkl x_ij              ▷ value change if k assigned to l
        if Δ < Δ'_y then
          k' ← k; l' ← l; Δ'_y ← Δ
        end if
      end for
    end for
    if Δ'_x ≤ Δ'_y then
      x_i'j' ← 1                               ▷ assign i' to j'
    else
      y_k'l' ← 1                               ▷ assign k' to l'
    end if
  end while
  return (x, y)
```

Proof. Consider the following BAP instance: C and D are zero matrices and elements of $2 \times 2 \times 3 \times 3$ matrix Q are all zero except $q_{1111} = -\epsilon, q_{1122} = q_{1133} = \epsilon, q_{2211} = q_{1123} = q_{1132} = 2\epsilon, q_{2222} = q_{2233} = L$, where ϵ and L are arbitrarily small and large positive numbers, respectively. At first the algorithm will assign $x_{11} = y_{11} = 1$, as q_{1111} is the smallest element in the array. Next, all indices $i, j \in M$ such that $i, j > 2$ and $k, l \in M$ such that $k, l > 3$ will be assigned within their respective groups. This is due to the fact that any assignment in those sets adds no additional cost to the current partial solution. Following that, $y_{22} = y_{33} = 1$ will be added. And finally, x_{22} will be set to 1 to complete a solution with the cost $3\epsilon + 2L$. However, an optimal solution in this case will contain $x_{11} = x_{22} = y_{11} = y_{23} = y_{32} = 1$ with an objective value of 5ϵ. Note that $\mathcal{A}(Q, C, D) = \frac{7\epsilon + 2L}{mn}$ and the result follows. □

We also consider a randomized version of *Greedy*, called ***GreedyRandomized***. In this variation a partial assignment is extended by a randomly picked x_{ij} or y_{kl} out of h best candidates (by solution value change), where h is some fixed number. Such approaches are generally called semi-greedy algorithms and form an integral part of many GRASP algorithms [70, 44]. To emphasize the randomized decisions in the algorithm and its linkages to GRASP, we call it *GreedyRandomized*.

Finally we discuss a construction heuristic based on rounding a fractional solution. In [38], a discretization procedure was introduced that computes a feasible solution to BAP with objective function value no more than that of the fractional solution. Given a fractional solution to BAP $(\mathbf{x}, \mathbf{y})$ (i.e. a solution to BAP (2.1)-(2.5) without integrality constrains (2.6)), we fix one side of the solution (say $\mathbf{x}$) and optimize $\mathbf{y}$ by solving a linear assignment problem to obtain a solution $\bar{\mathbf{y}}$. Then, fix $\bar{\mathbf{y}}$ and solve a linear assignment problem to find a solution $\bar{\mathbf{x}}$. Output the solution $(\bar{\mathbf{x}}, \bar{\mathbf{y}})$ as a result. We denote this approach as ***Rounding***.

Theorem 19. *A feasible solution* $(\mathbf{x}^*, \mathbf{y}^*)$ *to BAP with the cost* $f(\mathbf{x}^*, \mathbf{y}^*) \leq \mathcal{A}(Q, C, D)$*, can be obtained in* $O(m^2n^2 + n^3)$ *time using the Rounding algorithm.*

Proof. Consider the fractional solution $(\mathbf{x}, \mathbf{y})$ where $x_{ij} = 1/m$ for all $i, j \in M$, and $y_{ij} = 1/n$ for all $i, j \in N$. Then $(\mathbf{x}, \mathbf{y})$ is a feasible solution to the relaxation of BAP obtained by removing the integrality restrictions (2.6). It is easy to see that $f(\mathbf{x}, \mathbf{y}) = \mathcal{A}(Q, C, D)$. One of the properties of *Rounding* discussed in [38] is that the resulting solution is no worse than the input fractional solution, in terms of objective value. Apply Rounding to $(\mathbf{x}, \mathbf{y})$ to obtain the desired solution. □

Rounding provides us with an alternative way to Corollary 15 for generating a BAP solution with objective value no worse than the average. Recall, that by Theorem 14 this solution is guaranteed to be no worse than $(m-1)!(n-1)!$ feasible solutions.

It should be noted that this discretization procedure could also be applied to BAP fractional solutions obtained from other sources, such as the solution to the relaxed version of an integer linear programming reformulation of BAP. Some of the linearization reformulations [79, 48, 94, 1] of the QAP can be modified to obtain the corresponding linearizations of BAP. Selecting only $\mathbf{x}$ and $\mathbf{y}$ part from continuous solutions and ignoring other variables in the linearization formulations can be used to initiate the rounding algorithm discussed above. However, in this case, the resulting solution is not guaranteed to be no worse than the average.

3.2 Neighborhood structures and properties

Let us now discuss various neighborhoods associated with a feasible solution of BAP and analyze their properties. We also consider worst case properties of a local optimum for these neighborhoods. All these neighborhoods are based on reassigning parts of $\mathbf{x} \in \mathcal{X}$, parts of $\mathbf{y} \in \mathcal{Y}$, or both. The neighborhoods that we consider can be classified into three categories: *h-exchange neighborhoods*, *$[h, p]$-exchange neighborhoods*, and *shift based neighborhoods*.

3.2.1 The h-exchange neighborhood

In this class of neighborhoods, we apply an h-exchange operation to $\mathbf{x}$ while keeping $\mathbf{y}$ unchanged or viceversa. Let us discuss this in detail with $h = 2$. The 2-exchange neighborhood is well studied in the QAP literature. Our version of 2-exchange for BAP is related to the QAP variation, but also have some significant differences due to the specific structure of our problem.

Let $(\mathbf{x}, \mathbf{y})$ be a feasible solution to BAP. Consider two elements $i_1, i_2 \in M$, $j_1, j_2 \in M'$, such that $x_{i_1 j_1} = x_{i_2 j_2} = 1$. Then the 2-*exchange* operation on the $\mathbf{x}$-variables produces $(\mathbf{x}', \mathbf{y})$, where $\mathbf{x}'$ is obtained from $\mathbf{x}$ by swapping assignments of i_1, i_2 and j_1, j_2 (i.e. setting $x_{i_1 j_2} = x_{i_2 j_1} = 1$ and $x_{i_1 j_1} = x_{i_2 j_2} = 0$). Let $\Delta^x_{i_1 i_2}$ be the change in the objective value from $(\mathbf{x}, \mathbf{y})$ to $(\mathbf{x}', \mathbf{y})$. I.e.,

$$
\begin{aligned}
\Delta^x_{i_1 i_2} =& f(\mathbf{x}', \mathbf{y}) - f(\mathbf{x}, \mathbf{y}) \\
=& \sum_{i=1}^{m} \sum_{j=1}^{m} \sum_{k=1}^{n} \sum_{l=1}^{n} q_{ijkl} x'_{ij} y_{kl} + \sum_{i=1}^{m} \sum_{j=1}^{m} c_{ij} x'_{ij} + \sum_{k=1}^{n} \sum_{l=1}^{n} d_{kl} y_{kl} \\
& - \sum_{i=1}^{m} \sum_{j=1}^{m} \sum_{k=1}^{n} \sum_{l=1}^{n} q_{ijkl} x_{ij} y_{kl} - \sum_{i=1}^{m} \sum_{j=1}^{m} c_{ij} x_{ij} - \sum_{k=1}^{n} \sum_{l=1}^{n} d_{kl} y_{kl} \\
=& \sum_{k=1}^{n} \sum_{l=1}^{n} (q_{i_1 j_2 kl} + q_{i_2 j_1 kl} - q_{i_1 j_1 kl} - q_{i_2 j_2 kl}) y_{kl} + c_{i_1 j_2} + c_{i_2 j_1} - c_{i_1 j_1} - c_{i_2 j_2}.
\end{aligned}
\tag{3.1}
$$

Let $2exchangeX(\mathbf{x}, \mathbf{y})$ be the set of all feasible solutions $(\mathbf{x}', \mathbf{y})$, obtained from $(\mathbf{x}, \mathbf{y})$ by applying the 2-exchange operation for all $i_1, i_2 \in M$ (with corresponding $j_1, j_2 \in M'$). Efficient computation of $\Delta^x_{i_1 i_2}$ is crucial in developing fast algorithms that use this neighborhood. For a fixed $\mathbf{y}$, consider the $m \times m$ matrix E such that $e_{ij} = \sum_{k=1}^{n} \sum_{l=1}^{n} q_{ijkl} y_{kl} + c_{ij}$. Then we can write $\Delta^x_{i_1 i_2} = e_{i_1 j_2} + e_{i_2 j_1} - e_{i_1 j_1} - e_{i_2 j_2}$. If the matrix E is available, this calculation can be done in constant time, and hence the neighborhood $2exchangeX(\mathbf{x}, \mathbf{y})$ can be explored in $O(m^2)$ time for an improving solution. Note that the values of E depend only on $\mathbf{y}$ and not on $\mathbf{x}$. Thus, we do not need to update E within a local search algorithm as long as $\mathbf{y}$ remains unchanged.

Likewise, we can define a 2-exchange operation on $\mathbf{y}$ by keeping $\mathbf{x}$ constant. Consider two elements $k_1, k_2 \in N$ and let l_1, l_2 be the corresponding assignments in N', such that $x_{k_1 l_1} = x_{k_2 l_2} = 1$. Then the 2-exchange operation will produce $(\mathbf{x}, \mathbf{y}')$, where $\mathbf{y}'$ is obtained from $\mathbf{y}$ by swapping assignments of k_1, k_2 and l_1, l_2 (i.e. setting $x_{k_1 l_2} = x_{k_2 l_1} = 1$ and $x_{k_1 l_1} = x_{k_2 l_2} = 0$). Let $\Delta^y_{k_1 k_2}$ be the change in the objective value from $(\mathbf{x}, \mathbf{y})$ to $(\mathbf{x}, \mathbf{y}')$. I.e.,

$$
\begin{aligned}
\Delta^y_{k_1 k_2} =& f(\mathbf{x}, \mathbf{y}') - f(\mathbf{x}, \mathbf{y}) \\
=& \sum_{i=1}^{m} \sum_{j=1}^{m} (q_{ijk_1 l_2} + q_{ijk_2 l_1} - q_{ijk_1 l_1} - q_{ijk_2 l_2}) x_{ij} + d_{k_1 l_2} + d_{k_2 l_1} - d_{k_1 l_1} - d_{k_2 l_2}.
\end{aligned}
\tag{3.2}
$$

Let $2exchangeY(\mathbf{x}, \mathbf{y})$ be the set of all feasible solutions $(\mathbf{x}, \mathbf{y}')$, obtained from $(\mathbf{x}, \mathbf{y})$ by applying the 2-exchange operation on $\mathbf{y}$ while keeping $\mathbf{x}$ unchanged. As in the previous case, efficient computation of $\Delta^y_{k_1k_2}$ is crucial in developing fast algorithms that use this neighborhood. For a fixed $\mathbf{x}$ consider an $n\times n$ matrix G such that $g_{kl} = \sum_{i=1}^m \sum_{j=1}^m q_{ijkl}x_{ij} + d_{kl}$. Then we can write $\Delta^y_{k_1k_2} = g_{k_1l_2} + g_{k_2l_1} - g_{k_1l_1} - g_{k_2l_2}$. If the matrix G is available, this calculation can be done in constant time and hence the neighborhood $2exchangeY(\mathbf{x}, \mathbf{y})$ can be explored in $O(n^2)$ time for an improving solution. Note that the values of G depends only on $\mathbf{x}$ and not on $\mathbf{y}$. Thus, we do not need to update G within a local search algorithm as long as $\mathbf{y}$ remains unchanged.

The *2-exchange neighborhood* of $(\mathbf{x}, \mathbf{y})$, denoted by $2exchange(\mathbf{x}, \mathbf{y})$, is given by

$$2exchange(\mathbf{x}, \mathbf{y}) = 2exchangeX(\mathbf{x}, \mathbf{y}) \cup 2exchangeY(\mathbf{x}, \mathbf{y}).$$

In a local search algorithm based on the $2exchange(\mathbf{x}, \mathbf{y})$ neighborhood, after each move, either $\mathbf{x}$ or $\mathbf{y}$ will be changed, but not both. To maintain our data structure, if $\mathbf{y}$ is changed, we update E in $O(m^2)$ time. More specifically, suppose a 2-exchange operation takes $(\mathbf{x}, \mathbf{y})$ to $(\mathbf{x}, \mathbf{y}')$, then E is updated as: $e_{ij} \leftarrow e_{ij} + q_{ijk_1l_2} + q_{ijk_2l_1} - q_{ijk_1l_1} - q_{ijk_2l_2}$, where $k_1, k_2 \in N, l_1, l_2 \in N'$ are the corresponding positions where the swap have occurred. Analogous changes will be performed on G in $O(n^2)$ time if $(\mathbf{x}, \mathbf{y})$ is changed to $(\mathbf{x}', \mathbf{y})$.

The general *h-exchange neighborhood* for BAP is obtained by replacing 2 in the above definition by $2, 3, \ldots, h$. Notice that the h-exchange neighborhood can be searched for an improving solution in $O(n^h)$ time, and already for $h = 3$, the running time of the algorithm that completely explores this neighborhood is $O(n^3)$. With the same asymptotic running time we could instead optimally reassign whole $\mathbf{x}$ (or $\mathbf{y}$) by solving the linear assignment problem with E (or G respectively) as the cost matrix. This fact suggests that any h larger that 3 potentially leads to a weaker algorithm in terms of running time. Such full reassignment can be viewed as a local search based on the special case of the h-exchange neighborhood with $h = n$. This special local search will be referred to as ***Alternating Algorithm*** and will be alternating between re-optimizing $\mathbf{x}$ and $\mathbf{y}$. For clarity, the pseudo code for this approach is presented in Algorithm 3. *Alternating Algorithm* is a strategy well-known in non-linear programming literature as *coordinate-wise descent.* Similar underlying ideas are used in the context of other bilinear programming problems by various authors [86, 76, 108].

Theorem 20. *The objective function value of a locally optimal solution for BAP based on the h-exchange neighborhood could be arbitrarily bad and could be worse than $\mathcal{A}(Q, C, D)$, for any h.*

Proof. For a small $\epsilon > 0$ and a large L, we consider BAP instance (Q, C, D) such that all of its cost elements are equal to 0, except $c_{11} = c_{22} = d_{11} = d_{22} = -\epsilon$, and $q_{1212} = -L$. Let a feasible solution $(\mathbf{x}, \mathbf{y})$ be such that $x_{11} = x_{22} = y_{11} = y_{22} = 1$. Then $(\mathbf{x}, \mathbf{y})$ is a local optimum for the h-exchange neighborhood. Note that this local optimum can only

Algorithm 3 *Alternating Algorithm*

Input: integers m, n; $m \times m \times n \times n$ array Q; feasible solution $(\mathbf{x}, \mathbf{y})$ to BAP
Output: feasible solution to BAP
while True **do**
$e_{ij} \leftarrow \sum_{k,l \in N} q_{ijkl} y_{kl} \; \forall i, j \in M$
$\mathbf{x}^* \leftarrow arg\, min_{\mathbf{x}' \in \mathcal{X}} \sum_{i,j \in M} e_{ij} x'_{ij}$ ▷ solving assignment problem for $\mathbf{x}$
$g_{kl} \leftarrow \sum_{i,j \in M} q_{ijkl} x^*_{ij} \; \forall k, l \in N$
$\mathbf{y}^* \leftarrow arg\, min_{\mathbf{y}' \in \mathcal{Y}} \sum_{k,l \in N} g_{kl} y'_{kl}$ ▷ solving assignment problem for $\mathbf{y}$
if $f(\mathbf{x}^*, \mathbf{y}^*) = f(\mathbf{x}, \mathbf{y})$ **then**
break
end if
$\mathbf{x} \leftarrow \mathbf{x}^*$; $\mathbf{y} \leftarrow \mathbf{y}^*$
end while
return $(\mathbf{x}, \mathbf{y})$

be improved by simultaneously making changes to both $\mathbf{x}$ and $\mathbf{y}$, which is not possible for this neighborhood. The objective function value of $(\mathbf{x}, \mathbf{y})$ is -4ϵ, while the optimal solution objective value is $-L$. □

Despite the negative result of Theorem 20, we will see in Section 3.3.3 that on average, 2-exchange and n-exchange (with *Alternating Algorithm*) are two of the most efficient neighborhoods to explore from a practical point of view. Moreover, when restricted to non-negative input array, we can establish some performance guarantees for 2-exchange (and consequently for any h-exchange) local search. In particular, we derive upper bounds on the local optimum solution value and the number of iterations to reach a solution not worse than this value bound. The proof technique follows [6], where authors obtained similar bounds for Koopmans-Beckman QAP.

Theorem 21. *For any BAP instance (Q, C, D) with non-negative Q and zero matrices C, D, the cost of the local optimum for the 2-exchange neighborhood is $f^* \leq \frac{2mn}{m+n}\mathcal{A}(Q, C, D)$.*

Proof. In this proof, for simplicity, we represent BAP as a permutation problem. As such, the permutation formulation of BAP is

$$\min_{\pi \in \Pi, \phi \in \Phi} \sum_{i=1}^{m} \sum_{k=1}^{n} q_{i\,\pi(i)\,k\,\phi(k)}, \tag{3.3}$$

where Π and Φ are sets of all permutations on $\{1, 2, \ldots, m\}$ and $\{1, 2, \ldots, n\}$, respectively. Cost of a particular permutation pair π, ϕ is $f(\pi, \phi) = \sum_{i=1}^{m} \sum_{k=1}^{n} q_{i\,\pi(i)\,k\,\phi(k)}$.

Let π_{ij} be the permutation obtained by applying a single 2-exchange operation to π on indices i and j. Define δ^{π}_{ij} as an objective value difference after applying such 2-exchange:

$$\delta^{\pi}_{ij}(\pi, \phi) = f(\pi_{ij}, \phi) - f(\pi, \phi) = \sum_{k=1}^{m} \left(q_{i\,\pi(j)\,k\,\phi(k)} + q_{j\,\pi(i)\,k\,\phi(k)} - q_{i\,\pi(i)\,k\,\phi(k)} - q_{j\,\pi(j)\,k\,\phi(k)} \right).$$

Similarly we can have ϕ_{kl} and δ^{ϕ}_{kl}:

$$\delta^{\phi}_{kl}(\pi,\phi) = f(\pi,\phi_{kl}) - f(\pi,\phi) = \sum_{i=1}^{n}\Big(q_{i\,\pi(i)\,k\,\phi(l)} + q_{i\,\pi(i)\,l\,\phi(k)} - q_{i\,\pi(i)\,k\,\phi(k)} - q_{i\,\pi(i)\,l\,\phi(l)}\Big)\,.$$

Summing up over all possible δ^{π}_{ij} and δ^{ϕ}_{kl} we get

$$\begin{aligned}\sum_{i,j=1}^{m}\delta^{\pi}_{ij}(\pi,\phi) &= \sum_{i,j=1}^{m}\sum_{k=1}^{n} q_{i\,\pi(j)\,k\,\phi(k)} + \sum_{i,j=1}^{m}\sum_{k=1}^{n} q_{j\,\pi(i)\,k\,\phi(k)} - \sum_{i,j=1}^{m}\sum_{k=1}^{n} q_{i\,\pi(i)\,k\,\phi(k)} - \sum_{i,j=1}^{m}\sum_{k=1}^{n} q_{j\,\pi(j)\,k\,\phi(k)}\\ &= 2\sum_{i,j=1}^{m}\sum_{k=1}^{n} q_{i\,\pi(j)\,k\,\phi(k)} - 2mf(\pi,\phi), \qquad (3.4)\end{aligned}$$

$$\sum_{k,l=1}^{n}\delta^{\phi}_{kl}(\pi,\phi) = 2\sum_{i=1}^{m}\sum_{k,l=1}^{n} q_{i\,\pi(i)\,k\,\phi(l)} - 2nf(\pi,\phi). \qquad (3.5)$$

Using (3.4) and (3.5) we can now compute an average cost change after 2-exchange operation on solution (π,ϕ).

$$\begin{aligned}\Delta(\pi,\phi) &= \frac{\sum_{i,j=1}^{m}\delta^{\pi}_{ij}(\pi,\phi) + \sum_{k,l=1}^{n}\delta^{\phi}_{kl}(\pi,\phi)}{m^2+n^2}\\ &= \frac{2\sum_{i,j=1}^{m}\sum_{k=1}^{n} q_{i\,\pi(j)\,k\,\phi(k)} + 2\sum_{i=1}^{m}\sum_{k,l=1}^{n} q_{i\,\pi(i)\,k\,\phi(l)} - 2(m+n)f(\pi,\phi)}{m^2+n^2}\\ &= \frac{2\sum_{i,j=1}^{m}\sum_{k=1}^{n} q_{i\,\pi(j)\,k\,\phi(k)} + 2\sum_{i=1}^{m}\sum_{k,l=1}^{n} q_{i\,\pi(i)\,k\,\phi(l)}}{m^2+n^2} - \lambda f(\pi,\phi) + \lambda\frac{2mn}{m+n}\mathcal{A} - \lambda\frac{2mn}{m+n}\mathcal{A}\\ &\le -\lambda(f(\pi,\phi) - \frac{2mn}{m+n}\mathcal{A}) + \mu - \lambda\frac{2mn}{m+n}\mathcal{A}, \qquad (3.6)\end{aligned}$$

where $\lambda = 2\frac{m+n}{m^2+n^2}$ and $\mu = \max_{\pi\in\Pi,\phi\in\Phi}\left[\frac{2\sum_{i,j=1}^{m}\sum_{k=1}^{n} q_{i\,\pi(j)\,k\,\phi(k)} + 2\sum_{i=1}^{m}\sum_{k,l=1}^{n} q_{i\,\pi(i)\,k\,\phi(l)}}{m^2+n^2}\right]$. Note that both λ and μ do not depend on any particular solution and are fixed for a given BAP instance.

We are ready to prove the theorem by contradiction. Let (π^*,ϕ^*) be the local optimum for 2-exchange local search, with the objective function cost $f^* = f(\pi^*,\phi^*)$. Assume now that $f(\pi^*,\phi^*) > \frac{2mn}{m+n}\mathcal{A}$. Then $-\lambda(f(\pi^*,\phi^*) - \frac{2mn}{m+n}\mathcal{A}) < 0$ and

$$
\begin{aligned}
\mu - \lambda\frac{2mn}{m+n}\mathcal{A} &= \max_{\pi\in\Pi,\phi\in\Phi}\left[\frac{2\sum_{i,j=1}^{m}\sum_{k=1}^{n} q_{i\,\pi(j)\,k\,\phi(k)} + 2\sum_{i=1}^{m}\sum_{k,l=1}^{n} q_{i\,\pi(i)\,k\,\phi(l)}}{m^2+n^2}\right] \\
&\quad - 2\frac{m+n}{m^2+n^2}\frac{2mn}{m+n}\frac{1}{mn}\sum_{i,j=1}^{m}\sum_{k,l=1}^{n} q_{ijkl} \\
&= \max_{\pi\in\Pi,\phi\in\Phi}\left[\frac{2\sum_{i,j=1}^{m}\sum_{k=1}^{n} q_{i\,\pi(j)\,k\,\phi(k)}}{m^2+n^2} + \frac{2\sum_{i=1}^{m}\sum_{k,l=1}^{n} q_{i\,\pi(i)\,k\,\phi(l)}}{m^2+n^2}\right] \\
&\quad - \frac{2\sum_{i,j=1}^{m}\sum_{k,l=1}^{n} q_{ijkl}}{m^2+n^2} - \frac{2\sum_{i,j=1}^{m}\sum_{k,l=1}^{n} q_{ijkl}}{m^2+n^2} \le 0, \qquad (3.7)
\end{aligned}
$$

which implies $\Delta(\pi^*, \phi^*) < 0$. As Δ is the average cost difference after applying 2-exchange, there exists some swap that decreases solution cost by at least $-\Delta(\pi^*, \phi^*)$, and that contradicts with (π^*, ϕ^*) being a local optimum. □

It is easy to see that the bound $\mu \le \lambda\frac{2mn}{m+n}\mathcal{A}$ from Theorem 21 is tight. Consider some arbitrary bilinear assignment (π, ϕ), and set all q_{ijkl} to zero except $q_{i\,\pi(i)\,k\,\phi(k)} = 1$, $\forall i\,\forall k$. Then $\mu = 4\dfrac{\sum_{i=1}^{m}\sum_{k=1}^{n} q_{i\,\pi(i)\,k\,\phi(k)}}{m^2+n^2} = \lambda\frac{2mn}{m+n}\mathcal{A} = \frac{4mn}{m^2+n^2}$.

Theorem 22. *For any BAP instance (Q, C, D) with elements of Q restricted to non-negative integers and zero matrices C, D, the local search algorithm that explores 2-exchange neighborhood will reach a solution with the cost at most $\frac{2mn}{m+n}\mathcal{A}(Q, C, D)$ in $O\left(\frac{m^2+n^2}{m+n}\log\sum q_{ijkl}\right)$ iterations.*

Proof. Inequality (3.6) can be also written as $\Delta(\pi, \phi) \le -\lambda f(\pi, \phi) + \mu$, and so any solution with $f(\pi, \phi) > \frac{\mu}{\lambda}$ would yield $\Delta(\pi, \phi) < 0$, and would have some 2-exchange improvement possible. Note that $\frac{2mn}{m+n}\mathcal{A} \ge \frac{\mu}{\lambda}$.

Consider a cost $f'(\pi, \phi) = f(\pi, \phi) - \frac{\mu}{\lambda}$. At every step of the 2-exchange local search $f'(\pi, \phi)$ is decreased by at least $\Delta(\pi, \phi)$ and becomes at most

$$f'(\pi, \phi) + \Delta(\pi, \phi) \le f'(\pi, \phi) + (-\lambda f(\pi, \phi) + \mu) = f'(\pi, \phi) - \lambda f'(\pi, \phi) = (1-\lambda) f'(\pi, \phi).$$

Since elements of Q are integer, the cost at each step must decrease by at least 1. Then a number of iterations t for $C'(\pi, \phi)$ to become less than or equal to zero has to satisfy

$$\begin{aligned} (1-\lambda)^{t-1}(f_{\max} - \frac{\mu}{\lambda}) - (1-\lambda)^{t}(f_{\max} - \frac{\mu}{\lambda}) &\geq 1, \\ (1-\lambda)^{t-1}(f_{\max} - \frac{\mu}{\lambda})(1-(1-\lambda)) &\geq 1, \\ (1-\lambda)^{t-1} &\geq \frac{1}{(f_{\max} - \frac{\mu}{\lambda})\lambda}, \\ (t-1)\log(1-\lambda) &\geq -\log\lambda(f_{\max} - \frac{\mu}{\lambda}), \\ t &\leq 1 + \frac{-\log\lambda(f_{\max} - \frac{\mu}{\lambda})}{\log(1-\lambda)}, \end{aligned} \tag{3.8}$$

where $f_{\max}$ is the highest possible solution value. It follows that

$$t \in O\left(\frac{1}{\lambda}\log\lambda(f_{\max} - \frac{\mu}{\lambda})\right) = O\left(\frac{m^2+n^2}{m+n}\log\frac{m+n}{m^2+n^2}(f_{\max} - \frac{\mu}{\lambda})\right). \tag{3.9}$$

This together with the fact that $f_{\max} - \frac{\mu}{\lambda} \leq f_{\max} \leq \sum_{i,j=1}^{m}\sum_{k,l=1}^{n} q_{ijkl}$ completes the proof. $\square$

It should be noted that the solution considered in the statement of Theorem 22 may not be a local optimum. The theorem simply states that, the solution of the desired quality will be reached by 2-exchange local search in polynomial time. It is known that for QAP, 2-exchange local search may sometimes reach local optimum in exponential number of steps [103].

In fact, these results can be obtained for the general QAP as well, by modifying the following proof accordingly.

Theorem 23. *For QAP with non-negative q_{ijkl} and the average solution value $\mathcal{A}$, the cost of the local optimum for the 2-exchange neighborhood is $C^* \leq \frac{2n(n-1)}{2n-1}\mathcal{A}$.*

Proof. The permutation formulation of QAP is

$$\min_{\pi\in\Pi} \sum_{i=1}^{n}\sum_{k=1}^{n} q_{i\,\pi(i)\,k\,\pi(k)},$$

where Π is the set of all permutations on $\{1, 2, \ldots, n\}$. Cost of a particular permutation π is

$$C(\pi) = \sum_{i=1}^{n}\sum_{k=1}^{n} q_{i\,\pi(i)\,k\,\pi(k)}.$$

Note that for QAP we can assume without loss of generality that $q_{ijil} = 0$ when $j \neq l$ and similarly $q_{ijkj} = 0$ when $i \neq k$. Also, by reassigning all $q_{ijkl} = q_{klij} = \frac{q_{ijkl}+q_{klij}}{2}$ we can make n^2 by n^2 matrix Q symmetric, while maintaining equivalence to the original problem.

We know that an average value of all solutions to a given QAP instance is computed as:

$$
\begin{aligned}
\mathcal{A} &= \frac{1}{n(n-1)} \sum_{i,j=1}^{n} \sum_{k \neq i} \sum_{l \neq j} q_{i\,j\,k\,l} + \frac{1}{n} \sum_{i,j=1}^{n} q_{i\,j\,i\,j} \\
&= \frac{1}{n(n-1)} \Big(\sum_{i,j,k,l=1}^{n} q_{i\,j\,k\,l} - \sum_{i,j=1}^{n} q_{i\,j\,i\,j} \Big) + \frac{1}{n} \sum_{i,j=1}^{n} q_{i\,j\,i\,j} \\
&= \frac{\sum_{i,j,k,l=1}^{n} q_{i\,j\,k\,l} + (n-2) \sum_{i,j=1}^{n} q_{i\,j\,i\,j}}{n(n-1)}.
\end{aligned}
$$

Let π_{ij} be the permutation obtained by applying 2-exchange to π on indices i and j. Define δ_{ij} as an objective value difference after applying such 2ex:

$$
\begin{aligned}
\delta_{ij}(\pi) = C(\pi_{ij}) - C(\pi)& \\
= \sum_{k \neq i,j} \big(& q_{i\,\pi(j)\,k\,\pi(k)} + q_{j\,\pi(i)\,k\,\pi(k)} - q_{i\,\pi(i)\,k\,\pi(k)} - q_{j\,\pi(j)\,k\,\pi(k)} \\
& + q_{k\,\pi(k)\,i\,\pi(j)} + q_{k\,\pi(k)\,j\,\pi(i)} - q_{k\,\pi(k)\,i\,\pi(i)} - q_{k\,\pi(k)\,j\,\pi(j)} \big) \\
& + q_{i\,\pi(j)\,i\,\pi(j)} + q_{i\,\pi(j)\,j\,\pi(i)} - q_{i\,\pi(i)\,i\,\pi(i)} - q_{i\,\pi(i)\,j\,\pi(j)} \\
& + q_{j\,\pi(i)\,j\,\pi(i)} + q_{j\,\pi(i)\,i\,\pi(j)} - q_{j\,\pi(j)\,j\,\pi(j)} - q_{j\,\pi(j)\,i\,\pi(i)} \\
= 2 \sum_{k \neq i,j} \big(& q_{i\,\pi(j)\,k\,\pi(k)} + q_{j\,\pi(i)\,k\,\pi(k)} - q_{i\,\pi(i)\,k\,\pi(k)} - q_{j\,\pi(j)\,k\,\pi(k)} \big) \\
& + q_{i\,\pi(j)\,i\,\pi(j)} + 2q_{i\,\pi(j)\,j\,\pi(i)} - q_{i\,\pi(i)\,i\,\pi(i)} - 2q_{i\,\pi(i)\,j\,\pi(j)} \\
& + q_{j\,\pi(i)\,j\,\pi(i)} - q_{j\,\pi(j)\,j\,\pi(j)} \\
= 2 \sum_{k=1}^{n} \big(& q_{i\,\pi(j)\,k\,\pi(k)} + q_{j\,\pi(i)\,k\,\pi(k)} - q_{i\,\pi(i)\,k\,\pi(k)} - q_{j\,\pi(j)\,k\,\pi(k)} \big) \\
& -2\, q_{i\,\pi(j)\,i\,\pi(i)} - 2q_{j\,\pi(i)\,i\,\pi(i)} + q_{i\,\pi(i)\,i\,\pi(i)} + 2q_{j\,\pi(j)\,i\,\pi(i)} \\
& -2\, q_{i\,\pi(j)\,j\,\pi(j)} - 2q_{j\,\pi(i)\,j\,\pi(j)} + q_{j\,\pi(j)\,j\,\pi(j)} \\
& + q_{i\,\pi(j)\,i\,\pi(j)} + 2q_{i\,\pi(j)\,j\,\pi(i)} \\
& + q_{j\,\pi(i)\,j\,\pi(i)}.
\end{aligned}
$$

$$
\begin{aligned}
\sum_{i,j=1}^{n} \delta_{ij}(\pi) \; &= 2 \sum_{i,j,k=1}^{n} q_{i\,\pi(j)\,k\,\pi(k)} + 2 \sum_{i,j,k=1}^{n} q_{j\,\pi(i)\,k\,\pi(k)} - 2 \sum_{i,j,k=1}^{n} q_{i\,\pi(i)\,k\,\pi(k)} - 2 \sum_{i,j,k=1}^{n} q_{j\,\pi(j)\,k\,\pi(k)} \\
&- 2 \sum_{i,j=1}^{n} q_{i\,\pi(j)\,i\,\pi(i)} - 2 \sum_{i,j=1}^{n} q_{j\,\pi(i)\,i\,\pi(i)} + \sum_{i,j=1}^{n} q_{i\,\pi(i)\,i\,\pi(i)} + 2 \sum_{i,j=1}^{n} q_{j\,\pi(j)\,i\,\pi(i)} \\
&- 2 \sum_{i,j=1}^{n} q_{i\,\pi(j)\,j\,\pi(j)} - 2 \sum_{i,j=1}^{n} q_{j\,\pi(i)\,j\,\pi(j)} + \sum_{i,j=1}^{n} q_{j\,\pi(j)\,j\,\pi(j)} \\
&+ \sum_{i,j=1}^{n} q_{i\,\pi(j)\,i\,\pi(j)} + 2 \sum_{i,j=1}^{n} q_{i\,\pi(j)\,j\,\pi(i)} \\
&+ \sum_{i,j=1}^{n} q_{j\,\pi(i)\,j\,\pi(i)} \\
&= 4 \sum_{i,j,k=1}^{n} q_{i\,\pi(j)\,k\,\pi(k)} - 4nC(\pi) - 8 \sum_{i=1}^{n} q_{i\,\pi(i)\,i\,\pi(i)} + 2n \sum_{i=1}^{n} q_{i\,\pi(i)\,i\,\pi(i)} + 2C(\pi) \\
&+ 2 \sum_{i,j=1}^{n} q_{i\,\pi(j)\,i\,\pi(j)} + 2 \sum_{i,j=1}^{n} q_{i\,\pi(j)\,j\,\pi(i)} \\
&= 4 \sum_{i,j,k=1}^{n} q_{i\,\pi(j)\,k\,\pi(k)} + 2(n-4) \sum_{i=1}^{n} q_{i\,\pi(i)\,i\,\pi(i)} \\
&+ 2 \sum_{i,j=1}^{n} q_{i\,\pi(j)\,i\,\pi(j)} + 2 \sum_{i,j=1}^{n} q_{i\,\pi(j)\,j\,\pi(i)} - 2(2n-1)C(\pi)
\end{aligned}
$$

We can now compute an average cost change after 2ex operation on given π.

$$
\begin{aligned}
\Delta(\pi) \; &= \frac{\sum_{i,j=1}^{n} \delta_{ij}(\pi)}{n^2} \\
&= \frac{4 \sum_{i,j,k=1}^{n} q_{i\,\pi(j)\,k\,\pi(k)} + 2(n-4) \sum_{i=1}^{n} q_{i\,\pi(i)\,i\,\pi(i)} + 2 \sum_{i,j=1}^{n} q_{i\,\pi(j)\,i\,\pi(j)} + 2 \sum_{i,j=1}^{n} q_{i\,\pi(j)\,j\,\pi(i)}}{n^2} \\
&- \frac{2(2n-1)}{n^2} C(\pi) + \frac{4n(n-1)}{n^2}\mathcal{A} - \frac{4n(n-1)}{n^2}\mathcal{A} \\
&\leq -\frac{2(2n-1)}{n^2}\left(C(\pi) - \frac{2n(n-1)}{2n-1}\mathcal{A}\right) + \frac{1}{n^2}(M - 4n(n-1)\mathcal{A}), \qquad (3.10)
\end{aligned}
$$

where $M = \max_{\pi\in\Pi}[4 \sum_{i,j,k=1}^{n} q_{i\,\pi(j)\,k\,\pi(k)} + 2(n-4) \sum_{i=1}^{n} q_{i\,\pi(i)\,i\,\pi(i)} + 2 \sum_{i,j=1}^{n} q_{i\,\pi(j)\,i\,\pi(j)} + 2 \sum_{i,j=1}^{n} q_{i\,\pi(j)\,j\,\pi(i)}]$.

We are ready to prove the theorem by contradiction. Let π^* be the local optimum permutation for 2ex local search, with the objective function cost $C^* = C(\pi^*)$. Assume now that $C(\pi^*) > \frac{2n(n-1)}{2n-1}\mathcal{A}$. Then $-\frac{2(2n-1)}{n^2}(C(\pi^*) - \frac{2n(n-1)}{2n-1}\mathcal{A}) < 0$ and

$$
\begin{aligned}
&M - 4n(n-1)\mathcal{A} \\
&= \max_{\pi\in\Pi}[4\sum_{i,j,k=1}^{n} q_{i\,\pi(j)\,k\,\pi(k)} + 2(n-4)\sum_{i=1}^{n} q_{i\,\pi(i)\,i\,\pi(i)} + 2\sum_{i,j=1}^{n} q_{i\,\pi(j)\,i\,\pi(j)} + 2\sum_{i,j=1}^{n} q_{i\,\pi(j)\,j\,\pi(i)}] \\
&\quad - 4\sum_{i,j,k,l=1}^{n} q_{i\,j\,k\,l} - 4(n-2)\sum_{i,j=1}^{n} q_{i\,j\,i\,j} \\
&\leq \max_{\pi\in\Pi}[(4\sum_{i,j,k=1}^{n} q_{i\,\pi(j)\,k\,\pi(k)} + 2\sum_{i,j=1}^{n} q_{i\,\pi(j)\,j\,\pi(i)} - 2\sum_{i=1}^{n} q_{i\,\pi(i)\,i\,\pi(i)}) - (4\sum_{i,j,k,l=1}^{n} q_{i\,\pi(j)\,k\,\pi(l)})] \\
&\quad + \max_{\pi\in\Pi}[(2(n-4)\sum_{i=1}^{n} q_{i\,\pi(i)\,i\,\pi(i)} + 2\sum_{i,j=1}^{n} q_{i\,\pi(j)\,i\,\pi(j)} + 2\sum_{i=1}^{n} q_{i\,\pi(i)\,i\,\pi(i)}) - (4(n-2)\sum_{i,j=1}^{n} q_{i\,\pi(j)\,i\,\pi(j)})] \\
&\leq 0,
\end{aligned}
$$

which implies $\Delta(\pi^*) < 0$. As Δ is the average cost difference after applying 2-exchange, there exists some swap that decreases solution cost by at least $-\Delta(\pi^*)$, and that contradicts π^* being a local optimum.

□

Theorem 24. *For QAP with non-negative integer valued q_{ijkl}, 2ex local search will reach a solution with the cost at most $\frac{2n(n-1)}{2n-1}\mathcal{A}$ in $O(n\log\sum q_{ijkl})$.*

Proof. Inequality (3.10) can be also written as $\Delta(\pi) \leq -\frac{2(2n-1)}{n^2}C(\pi) + \frac{M}{n^2}$, and so any solution with $C(\pi) > \frac{M}{2(2n-1)}$ would yield $\Delta(\pi) < 0$, and would have some 2ex improvement possible. Note that $\frac{2n(n-1)}{2n-1}\mathcal{A} \geq \frac{M}{2(2n-1)}$.

Consider a cost $C'(\pi) = C(\pi) - \frac{M}{2(2n-1)}$. At every step of the 2ex local search $C'(\pi)$ becomes $C(\pi) - \frac{M}{2(2n-1)} + \Delta(\pi) \leq C(\pi) - \frac{M}{2(2n-1)} + (-\frac{2(2n-1)}{n^2}C(\pi) + \frac{M}{n^2}) = (C(\pi) - \frac{M}{2(2n-1)})(1 - \frac{2(2n-1)}{n^2}) = (1 - \frac{2(2n-1)}{n^2})C'(\pi)$. Also, let $C'_{\max} = C_{\max} - \frac{M}{2(2n-1)}$, where $C_{\max}$ is the solution with the highest possible cost. Since elements of Q are integer, the cost at each step must decrease by at least 1. Then a number of iterations t for $C'(\pi)$ to become less than or equal to zero has to satisfy

$$
\begin{aligned}
(1-\frac{2(2n-1)}{n^2})^{t-2}C'_{\max} - (1-\frac{2(2n-1)}{n^2})^{t-1}C'_{\max} &\geq 1,\\
(1-\frac{2(2n-1)}{n^2})^{t-2}C'_{\max}(1-(1-\frac{2(2n-1)}{n^2})) &\geq 1,\\
(1-\frac{2(2n-1)}{n^2})^{t-2} &\geq \frac{1}{C'_{\max}\frac{2(2n-1)}{n^2}},\\
(t-2)\log{(1-\frac{2(2n-1)}{n^2})} &\geq -\log\frac{2(2n-1)}{n^2}C'_{\max},\\
t \leq 2 + \frac{-\log\frac{2(2n-1)}{n^2}C'_{\max}}{\log{(1-\frac{2(2n-1)}{n^2})}} &\in O(n\log\frac{1}{n}C'_{\max}.
\end{aligned}
$$

This together with the fact that $C'_{\max} \leq C_{\max} \leq \sum_{i,j=1}^{m}\sum_{k,l=1}^{n} q_{ijkl}$ completes the proof. □

3.2.2 [*h*,*p*]-exchange neighborhoods

Recall that in the h-exchange neighborhood we change either the $\mathbf{x}$ variables or the $\mathbf{y}$ variables, but not both. Simultaneous changes in $\mathbf{x}$ and $\mathbf{y}$ could lead to more powerful neighborhoods, but with additional computational effort in exploring them. With this motivation, we introduce the *$[h,p]$-exchange neighborhood* for BAP.

In the $[h,p]$-exchange neighborhood, for each h-exchange operation on $\mathbf{x}$ variables, we consider all possible p-exchange operations on $\mathbf{y}$ variables. Thus, the $[h,p]$-exchange neighborhood is the set of all solutions $(\mathbf{x}',\mathbf{y}')$ obtained from the given solution $(\mathbf{x},\mathbf{y})$, such that $\mathbf{x}'$ differs from $\mathbf{x}$ in at most h assignments, and $\mathbf{y}'$ differs from $\mathbf{y}$ in at most p assignments. The size of this neighborhood is $\Theta(m^h n^p)$.

Theorem 25. *The objective function value of a locally optimal solution for the $[h,p]$-exchange neighborhood could be arbitrarily bad. If $h < \frac{m}{2}$ or $p < \frac{n}{2}$ this value could be arbitrarily worse than $\mathcal{A}(Q,C,D)$.*

Proof. Let $\epsilon > 0$ be an arbitrarily small and L be an arbitrarily large numbers. Consider the BAP instance (Q,C,D) such that all of the associated cost elements are equal to 0, except $q_{iikk} = -\epsilon, q_{i(i+1 \mod m)k(k+1 \mod n)} = -L, q_{iik(k+1 \mod n)} = \frac{hL}{m-h} \quad \forall i \in M\,\forall k \in N$. Let $(\mathbf{x},\mathbf{y})$ be a feasible solution such that $x_{ii} = 1 \quad \forall i \in M$ and $y_{kk} = 1 \quad \forall k \in N$. Note that $f(\mathbf{x},\mathbf{y}) = -mn\epsilon$.

We first show that $(\mathbf{x},\mathbf{y})$ is a local optimum for the $[h,p]$-exchange neighborhood. If we assume the opposite and $(\mathbf{x},\mathbf{y})$ is not a local optimum, then there exist a solution $(\mathbf{x}',\mathbf{y}')$ with $\mathbf{x}'$ being different from $\mathbf{x}$ in at most h assignments, $\mathbf{y}'$ being different from $\mathbf{y}$ in at most p assignments, and $f(\mathbf{x}',\mathbf{y}') - f(\mathbf{x},\mathbf{y}) < 0$. Since the summation for $f(\mathbf{x},\mathbf{y})$ comprised of exactly mn elements of Q with value $-\epsilon$, the only way to get an improving solution is to get some number of elements with value $-L$, and therefore to flip some number of x_{ii} to $x_{i(i+1 \mod m)}$ and y_{kk} to $y_{k(k+1 \mod n)}$. Let $1 < u \leq h$ and $1 < v \leq p$ be the number of such elements $u = |\{i \in M | x'_{i(i+1 \mod m)} = 1\}|$ and $v = |\{k \in N | y'_{k(k+1 \mod n)} = 1\}|$ in $(\mathbf{x}', \mathbf{y}')$. Then we know that the cost function $f(\mathbf{x}', \mathbf{y}')$ contains exactly uv number of $-L$. However, each of the v elements of type $y'_{k(k+1 \mod n)} = 1$ also contributes at least $(m-h)\frac{hL}{m-h} = hL$ to the objective value (due to remaining $m-h$ elements of type $x_{ii} = 1$ being unchanged). From this we get that $f(\mathbf{x}', \mathbf{y}') > mn(-\epsilon) + uv(-L) + hv(L) = f(\mathbf{x}, \mathbf{y}) + vL(h-u)$, and since $u \leq h$ we get $f(\mathbf{x}', \mathbf{y}') - f(\mathbf{x}, \mathbf{y}) > 0$ which contradicts the fact that $(\mathbf{x}', \mathbf{y}')$ is an improving solution to $(\mathbf{x}, \mathbf{y})$. Hence, $(\mathbf{x}, \mathbf{y})$ must be a local optimum.

We also get that an optimal solution for this instance is $x_{i(i+1 \mod m)} = 1 \quad \forall i \in M$ and $y_{k(k+1 \mod n)} = 1 \quad \forall k \in N$ with a total cost of $-mnL$. The average value of all feasible solutions is $\mathcal{A}(Q, C, D) = \dfrac{mn(-L) + mn(-\epsilon) + mn\frac{hL}{m-h}}{mn} = L\frac{2h-m}{m-h} - \epsilon$. $h < \frac{m}{2}$ and appropriate choice of ϵ, L guarantee us that considered local optimum is arbitrarily worse than $\mathcal{A}(Q, C, D)$. The construction of the example for the case $p < \frac{n}{2}$ is similar, so we omit the details. $\square$

One particular case of the $[h, p]$-exchange neighborhood deserves a special mention. If $p = n$, then for each candidate h-exchange solution $\mathbf{x}'$ we will consider all possible assignments for $\mathbf{y}$. To find the optimal $\mathbf{y}$ given $\mathbf{x}'$, we can solve a linear assignment problem with cost matrix $g_{kl} = \sum_{i=1}^{m} \sum_{j=1}^{m} q_{ijkl} x'_{ij} + d_{kl}$, as in the *Alternating Algorithm*. Analogous situation appears when we consider $[h, p]$-exchange neighborhood with $h = m$.

A set of solutions defined by the union of $[h, n]$-exchange and $[m, p]$-exchange neighborhoods, for the case $h = p$, will be called simply *optimized h-exchange neighborhood*. Note that the optimized h-exchange neighborhood is exponential in size, but it can be searched in $O(m^h n^3 + n^h m^3)$ time due to the fact that for fixed $\mathbf{x}$ ($\mathbf{y}$), optimal $f(\mathbf{x}, \mathbf{y}')$ ($f(\mathbf{x}', \mathbf{y})$) can be found in $O(n^3)$ time. Neighborhoods similar to optimized 2-exchange were used for unconstrained bipartite binary quadratic program by Glover et al. [54], and for the bipartite quadratic assignment problem by Punnen and Wang [109].

As in the case of h-exchange, some performance bounds for optimized h-exchange neighborhood can be established, if the input array Q is not allowed to have negative elements.

Theorem 26. *There exists a solution with the cost $f \leq (m+n)\mathcal{A}(Q, C, D)$ in the optimized 2-exchange neighborhood of every solution to BAP, for any instance (Q, C, D) with non-negative Q and zero matrices C, D.*

Proof. The proof will follow the structure of Theorem 21, and will focus on the average solution change to a given permutation pair solution (π, ϕ) to BAP.

Let π_{ij} be the permutation obtained by applying a single 2-exchange operation to π on indices i and j, and ϕ^* be the optimal permutation that minimizes the solution cost for such fixed π_{ij}. Define δ^{π}_{ij} as the objective value difference after applying such operation:

$$\delta^{\pi}_{ij}(\pi,\phi) = f(\pi_{ij},\phi^*) - f(\pi,\phi) = \sum_{u=1}^{m}\sum_{k=1}^{n} q_{u\,\pi_{ij}(u)\,k\,\phi^*(k)} - f(\pi,\phi) \le \frac{1}{n}\sum_{u=1}^{m}\sum_{k,l=1}^{n} q_{u\,\pi_{ij}(u)\,k\,l} - f(\pi,\phi).$$

The last inequality due to the fact that, for fixed π_{ij}, the value of the solution with the optimal ϕ^* is not worse than the average value of all such solutions. We also know that for any $k,l \in N$,

$$\sum_{u=1}^{m} q_{u\,\pi_{ij}(u)\,k\,l} = \sum_{u=1}^{m} q_{u\,\pi(u)\,k\,l} + q_{i\,\pi(j)\,k\,l} + q_{j\,\pi(i)\,k\,l} - q_{i\,\pi(i)\,k\,l} - q_{j\,\pi(j)\,k\,l}.$$

and, therefore,

$$\delta^{\pi}_{ij}(\pi,\phi) \le \frac{1}{n}\sum_{k,l=1}^{n}\sum_{u=1}^{m} q_{u\,\pi(u)\,k\,l} + \frac{1}{n}\sum_{k,l=1}^{n}\Big(q_{i\,\pi(j)\,k\,l} + q_{j\,\pi(i)\,k\,l} - q_{i\,\pi(i)\,k\,l} - q_{j\,\pi(j)\,k\,l}\Big) - f(\pi,\phi).$$

Analogous result can be derived for similarly defined δ^{ϕ}_{kl}:

$$\delta^{\phi}_{kl}(\pi,\phi) \le \frac{1}{m}\sum_{i,j=1}^{m}\sum_{v=1}^{n} q_{i\,j\,v\,\phi(v)} + \frac{1}{m}\sum_{i,j=1}^{m}\Big(q_{i\,j\,k\,\phi(l)} + q_{i\,j\,l\,\phi(k)} - q_{i\,j\,k\,\phi(k)} - q_{i\,j\,l\,\phi(l)}\Big) - f(\pi,\phi).$$

We can now get an upper bound on the average cost change after optimized 2-exchange operation on solution (π,ϕ).

$$\begin{aligned}
\Delta(\pi,\phi) &= \frac{\sum_{i,j=1}^{m}\delta^{\pi}_{ij}(\pi,\phi) + \sum_{k,l=1}^{n}\delta^{\phi}_{kl}(\pi,\phi)}{m^2+n^2}\\
&\le \frac{\frac{m^2}{n}\sum_{u=1}^{m}\sum_{k,l=1}^{n} q_{u\,\pi(u)\,k\,l} + \frac{2}{n}\sum_{i,j=1}^{m}\sum_{k,l=1}^{n} q_{i\,j\,k\,l} - \frac{2m}{n}\sum_{i=1}^{m}\sum_{k,l=1}^{n} q_{i\,\pi(i)\,k\,l} - m^2 f(\pi,\phi)}{m^2+n^2}\\
&\quad + \frac{\frac{n^2}{m}\sum_{i,j=1}^{m}\sum_{v=1}^{n} q_{i\,j\,v\,\phi(v)} + \frac{2}{m}\sum_{i,j=1}^{m}\sum_{k,l=1}^{n} q_{i\,j\,k\,l} - \frac{2n}{m}\sum_{i,j=1}^{m}\sum_{k=1}^{n} q_{i\,j\,k\,\phi(k)} - n^2 f(\pi,\phi)}{m^2+n^2}\\
&= \frac{(m^3-2m^2)\sum_{i=1}^{m}\sum_{k,l=1}^{n} q_{i\,\pi(i)\,k\,l} + (n^3-2n^2)\sum_{i,j=1}^{m}\sum_{v=1}^{n} q_{i\,j\,v\,\phi(v)}}{mn(m^2+n^2)}\\
&\quad + \frac{2(m+n)\sum_{i,j=1}^{m}\sum_{k,l=1}^{n} q_{i\,j\,k\,l}}{mn(m^2+n^2)} - f(\pi,\phi)\\
&\le \mu - f(\pi,\phi),
\end{aligned}$$

where

$$\mu = \max_{\pi\in\Pi,\phi\in\Phi}\left[\frac{m^3\sum_{i=1}^{m}\sum_{k,l=1}^{n} q_{i\,\pi(i)\,k\,l} + n^3\sum_{i,j=1}^{m}\sum_{v=1}^{n} q_{i\,j\,v\,\phi(v)} + 2(m+n)\sum_{i,j=1}^{m}\sum_{k,l=1}^{n} q_{ijkl}}{mn(m^2+n^2)}\right].$$

Note that μ does not depend on any particular solution and is fixed for a given BAP instance.

For any given solution (π, ϕ) to BAP, either $f(\pi, \phi) \leq \mu$ or $f(\pi, \phi) > \mu$, which means that $\Delta(\pi, \phi) \leq 0$, and so there exists an optimized 2-exchange operation that improves our solution cost by at least $f(\pi, \phi) - \mu$, thus, making it not worse than μ. We also notice that,

$$\begin{aligned} \mu - (m+n)\mathcal{A} &= \mu - \frac{m+n}{mn} \sum_{i,j=1}^{m} \sum_{k,l=1}^{n} q_{ijkl} = \mu - \frac{(m+n)(m^2+n^2)}{mn(m^2+n^2)} \sum_{i,j=1}^{m} \sum_{k,l=1}^{n} q_{ijkl} \\ &= \max_{\pi \in \Pi} \left[\frac{m^3 \sum_{i=1}^{m} \sum_{k,l=1}^{n} q_{i\,\pi(i)\,k\,l}}{mn(m^2+n^2)} \right] - \frac{m^3 \sum_{i,j=1}^{m} \sum_{k,l=1}^{n} q_{ijkl}}{mn(m^2+n^2)} \\ &+ \max_{\phi \in \Phi} \left[\frac{n^3 \sum_{i,j=1}^{m} \sum_{v=1}^{n} q_{i\,j\,v\,\phi(v)}}{mn(m^2+n^2)} \right] - \frac{n^3 \sum_{i,j=1}^{m} \sum_{k,l=1}^{n} q_{ijkl}}{mn(m^2+n^2)} \\ &+ \frac{2(m+n) \sum_{i,j=1}^{m} \sum_{k,l=1}^{n} q_{ijkl}}{mn(m^2+n^2)} - \frac{(m^2 n + n^2 m) \sum_{i,j=1}^{m} \sum_{k,l=1}^{n} q_{ijkl}}{mn(m^2+n^2)} \leq 0, \end{aligned} \tag{3.11}$$

and so $(m+n)\mathcal{A} \geq \mu$, which completes the proof. $\square$

We now show that by exploiting the properties of optimized h-exchange neighborhood, one can obtain a solution with an improved domination number, compared to the result in Theorem 14.

Theorem 27. *For an integer h, a feasible solution to BAP, which is no worse than $\Omega((m-1)!(n-1)! + m^h n! + n^h m!)$ feasible solutions, can be found in $O(m^h n^3 + n^h m^3)$ time.*

Proof. We show that the solution described in the statement of the theorem, can be obtained in the desired running time by choosing the best solution in the optimized h-exchange neighborhood of a solution with objective function value no worse than $\mathcal{A}(Q, C, D)$.

Let $(\mathbf{x}^*, \mathbf{y}^*) \in \mathcal{F}$ be a BAP solution such that $f(\mathbf{x}^*, \mathbf{y}^*) \leq \mathcal{A}(Q, C, D)$. Solution like that can be found in $O(m^2 n^2)$ time using Corollary 15. From the proof of Theorem 14 we know that there exists a set $R_\sim$ of $(m-1)!(n-1)!$ solutions, with one solution from every class defined by the equivalence relation $\sim$, such that $f(\mathbf{x}, \mathbf{y}) \geq \mathcal{A}(Q, C, D) \geq f(\mathbf{x}^*, \mathbf{y}^*)$ for every $(\mathbf{x}, \mathbf{y}) \in R_\sim$. Let R_x denote the $[h, n]$-exchange neighborhood of $(\mathbf{x}^*, \mathbf{y}^*)$, and let R_y denote the $[m, h]$-exchange neighborhood of $(\mathbf{x}^*, \mathbf{y}^*)$. Note that $R_x \cup R_y$ is the optimized h-exchange neighborhood of $(\mathbf{x}^*, \mathbf{y}^*)$. $R_x \cup R_y$ can be searched in $O(m^h n^3 + n^h m^3)$ time, and the result of the search has the objective function value less or equal than every $(\mathbf{x}, \mathbf{y}) \in R_\sim \cup R_x \cup R_y$.

Consider $R'_x \subset R_x$ ($R'_y \subset R_y$) to be the set of solutions constructed in the same way as R_x (R_y), but now only considering those reassignments of h-sets $S \in M$ ($S \in N$) that are different from $\mathbf{x}^*$ ($\mathbf{y}^*$) on entire S. By simple enumerations it can be shown that $|R'_x| = \binom{m}{h}(!h)n!$, $|R'_y| = \binom{n}{h}(!h)m!$ and $|R'_x \cap R'_y| = \binom{m}{h}(!h)\binom{n}{h}(!h)$, where $!h$ denotes the number of derangements (i.e. permutations without fixed points) of h elements. Furthermore, $|R_\sim \cap$

$R'_x| \leq \binom{m}{h}(!h)(n-1)!$ and $|R_\sim \cap R'_y| \leq \binom{n}{h}(!h)(m-1)!$. The later two inequalities are due to the fact that for some fixed $\mathbf{x}'$ ($\mathbf{y}'$), the relation $\sim$ partitions the set of solutions $\{\mathbf{x}'\} \times \mathcal{Y}$ ($\mathcal{X} \times \{\mathbf{y}'\}$) into equivalence classes of size n (m) exactly, and each such class contains at most one element of $R_\sim$. Now we get that

$$
\begin{aligned}
|R_\sim \cup R_x \cup R_y| &\geq |R_\sim \cup R_x^d \cup R_y^d| \\
&\geq |R_\sim| + |R_x^d| + |R_y^d| - |R_\sim \cap R_x^d| - |R_\sim \cap R_y^d| - |R_x^d \cap R_y^d| \\
&\geq (m-1)!(n-1)! + \binom{m}{h}(!h)n! + \binom{n}{h}(!h)m! \\
&\quad - \binom{m}{h}(!h)(n-1)! - \binom{n}{h}(!h)(m-1)! - \binom{m}{h}(!h)\binom{n}{h}(!h) \\
&\in \Omega((m-1)!(n-1)! + m^h n! + n^h m!),
\end{aligned}
$$

which concludes the proof. $\square$

3.2.3 Shift based neighborhoods

Following the equivalence class example in Section 2.3.3, the *shift* neighborhood of a given solution $(\mathbf{x}, \mathbf{y})$ will be comprised of all m solutions $(\mathbf{x}', \mathbf{y})$, such that $x'_{ij} = x_{i(j+a \mod m)}, \forall a \in M$ and all n solutions $(\mathbf{x}, \mathbf{y}')$, such that $y'_{kl} = y_{k(l+b \mod m)}, \forall b \in N$. Alternatively, shift neighborhood can be described in terms of the permutation formulation of BAP. Given a permutation pair (π, ϕ), we are looking at all m solutions (π', ϕ), such that $\pi'(i) = \pi(i) + a \mod m, \forall a \in M$, and all n solutions (π, ϕ'), such that $\phi'(k) = \phi(k) + b \mod m, \forall b \in N$. Intuitively this means that, either π will be cyclically shifted by a or ϕ will be cyclically shifted by b, hence the name of this neighborhood. An iteration of the local search algorithm based on Shift neighborhood will take $O(mn^2)$ time, as we are required to fully recompute each of the m (resp. n) solutions objective values.

Using the same asymptotic running time per iteration, it is possible to explore the neighborhood of a larger size, with the help of additional data structures e_{ij}, g_{kl} (see Section 3.2.1) that maintain partial sums of assigning $i \in M$ to $j \in M'$ and $k \in N$ to $l \in N'$ given $\mathbf{y}$ and $\mathbf{x}$ respectively. Consider $\Theta(n^2)$ size neighborhood *shift+shuffle* defined as follows. For a given permutation solution (π, ϕ) this neighborhood will contain all (π', ϕ) such that

$$
\pi'(i) = \pi\left((i \mod \lfloor \frac{m}{u} \rfloor)u + \lfloor \frac{i}{\lfloor \frac{m}{u} \rfloor} \rfloor + a \mod m\right), \quad \forall a \in M, \forall u \in \{1, 2, \ldots, \lfloor \frac{m}{2} \rfloor\}, \tag{3.12}
$$

and all (π, ϕ') such that

$$
\phi'(k) = \phi\left((k \mod \lfloor \frac{n}{v} \rfloor)v + \lfloor \frac{k}{\lfloor \frac{n}{v} \rfloor} \rfloor + b \mod n\right), \quad \forall b \in N, \forall v \in \{1, 2, \ldots, \lfloor \frac{n}{2} \rfloor\}. \tag{3.13}
$$

Two of the above equations are sufficient for the case of $m \mod u = 0$ or $n \mod v = 0$. Otherwise, for all $i > m-(m \mod u)$ and all $k > n-(n \mod v)$ an arbitrary reassignment could be applied (for example $\pi'(i) = \pi(i)$ and $\phi'(k) = \phi(k)$). One can visualize shuffle operation as splitting elements of a permutation into buckets of the same size (u or v in the formulas above), and then forming a new permutation by placing first elements from each bucket in the beginning, followed by second elements of each bucket, and so on. Figure 3.1 depicts such shuffling for a permutation π. By combining shift and shuffle we increase the

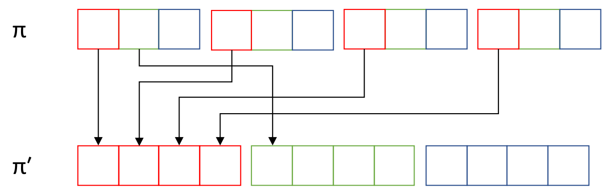

Figure 3.1: Example of shuffle operation on permutation π, with $u = 3$

size of the explored neighborhood, at no extra asymptotic running time cost for the local search implementations.

Local search algorithms that explore shift or shift+shuffle neighborhoods could potentially be stuck in the arbitrarily bad local optimum, following the same argument as in Theorem 20.

If we allow applying shift simultaneously to both $\mathbf{x}$ and $\mathbf{y}$ we will consider all mn neighbors of the current solution, precisely as in equivalence class example from Section 2.3.3. We will call this *dual shift* neighborhood of a solution $(\mathbf{x}, \mathbf{y})$. Notice that a local search algorithm that explores this neighborhood reaches a local optimum only after a single iteration, with running time $O(m^2n^2)$.

A much larger *optimized shift* neighborhood will be defined as follows. For every shift operation on $\mathbf{x}$ we consider all possible assignments of $\mathbf{y}$, and vice versa, for each shift on $\mathbf{y}$ we will consider all possible assignments of $\mathbf{x}$. Just like in the case of optimized h-exchange, this neighborhood is exponential in size, but can be efficiently explored in $O(mn^3)$ running time by solving corresponding linear assignment problems.

Theorem 28. *For local search based on dual shift and optimized shift neighborhoods, the final solution value is guaranteed to be no worse than $\mathcal{A}(Q, C, D)$.*

Proof. The proof for dual shift neighborhood follows from the fact that we are completely exploring the equivalence class defined by $\sim$ of a given solution, as in Corollary 15.

For optimized shift, notice that for each shift on one side of $(\mathbf{x}, \mathbf{y})$ we consider all possible solutions on the other side. This includes all possible shifts on that respective side.

Therefore the set of solutions of optimized shift neigborhood includes the set of solutions of dual shift neighborhood, and contains the solution with the value at most $\mathcal{A}(Q, C, D)$. □

In [38] we have explored the complexity of a special case of BAP where Q, observed as a $m^2 \times n^2$ matrix, is restricted to be of a fixed rank. The rank of such Q is said to be at most r if and only if there exist some $m \times m$ matrices $A^p = (a_{ij}^p)$ and $n \times n$ matrices $B^p = (b_{ij}^p)$, $p = 1, \ldots, r$, such that

$$q_{ijkl} = \sum_{p=1}^{r} a_{ij}^p b_{kl}^p \tag{3.14}$$

for all $i, j \in M$, $k, l \in N$.

Theorem 29. *Alternating Algorithm and local search algorithms that explore optimized h-exchange and optimized shift neighborhoods will find an optimal solution to BAP (Q, C, D), if Q is a non-negative matrix of rank 1, and both C and D are zero matrices.*

Proof. Note that in the case described in the statement of the theorem, we are looking for such $(\mathbf{x}^*, \mathbf{y}^*)$ that minimizes $(\sum_{i,j=1}^{m} a_{ij} x_{ij}^*) \cdot (\sum_{k,l=1}^{n} b_{kl} y_{kl}^*)$, where $q_{ijkl} = a_{ij} b_{kl}$, $\forall i, j \in M$, $k, l \in N$. If we are restricted to non-negative numbers, solutions to corresponding linear assignment problems would be an optimal solution to this BAP. It is easy to see that, for any fixed $\mathbf{x}$, a solution of the smallest value will be produced by $\mathbf{y}^*$. And viceversa, for any fixed $\mathbf{y}$, a solution of the smallest value will be produced by $\mathbf{x}^*$.

Optimized h-exchange neighborhood, optimized shift neighborhood and the neighborhood that *Alternating Algorithm* is based on, all contain the solution that has one side of $(\mathbf{x}, \mathbf{y})$ unchanged and has the optimal assignment on the other side. Therefore, the local search algorithms that explore these neighborhoods will proceed to find optimal $(\mathbf{x}^*, \mathbf{y}^*)$ in at most 2 iterations. □

3.3 Experimental analysis

3.3.1 Experimental design and test problems

In this section we present general information on the design of our experiments and generation of test problems.

All experiments are conducted on a PC with Intel Core i7-4790 processor, 32 GB of memory under control of Linux Mint 17.3 (Linux Kernel 3.19.0-32-generic) 64-bit operating system. Algorithms are coded using Python 2.7 programming language and run via PyPy 5.3 implementation of Python. The linear assignment problem, that appears as a subproblem for several algorithms, is solved using Hungarian algorithm [90] implementation in Python.

Test problems

As there are no existing benchmark instances available for BAP, we have created several sets of test problems, which could be used by other researchers in the future experimental analysis. Three categories of problem instances are considered: ***uniform***, ***normal*** and ***euclidean***.

- For *uniform* instances we set $c_{ij}, d_{kl} = 0$ and the values q_{ijkl} are generated randomly with uniform distribution from the interval $[0, mn]$ and rounded to the nearest integer.
- For *normal* instances we set $c_{ij}, d_{kl} = 0$ and the values q_{ijkl} are generated randomly following normal distribution with mean $\mu = \frac{mn}{2}$, standard deviation $\sigma = \frac{mn}{6}$ and rounded to the nearest integer.
- For *euclidean* instances we generate randomly with uniform distribution four sets of points A, B, U, V in Euclidean plane of size $[0, 1.5\sqrt[2]{mn}] \times [0, 1.5\sqrt[2]{mn}]$, such that $|A| = |B| = m$, $|U| = |V| = n$. Then C and D are chosen as zero vectors, and $q_{ijkl} = ||a_i - u_k|| \cdot ||b_j - v_l||$ (rounded to the nearest integer), where $a_i \in A, b_j \in B, u_k \in U, v_l \in V$.

Test problems are named using the convention "type size number", where type $\in$ {*uniform*, *normal*, *euclidean*}, size is of the form $m \times n$, and number $\in \{0, 1, \ldots\}$. For every instance type and size we have generated 10 problems, and all the results of experiments will be averaged over those 10 problems. For example, in a table or a figure, a data point for "uniform 50×50" would be the average among the 10 generated instances. This applies to objective function values, running times and number of iterations, and would not be explicitly mentioned throughout the rest of the paper. Problem instances, results for our final set of experiments as well as best found solutions for every instance are available upon request from Abraham Punnen (apunnen@sfu.ca).

3.3.2 Experimental analysis of construction heuristics

In Section 3.1 we presented several construction approaches to generate a solution to BAP. In this section we discuss results of computational experiments using these heuristics.

The experimental results are summarized in Table 3.1. For the heuristic *GreedyRandomized*, we have considered the candidate list size 2, 4 and 6. In the table, columns GreedyRandomized2 and GreedyRandomized4 refer to implementations with candidate list size of 2 and 4, respectively. Results for candidate list size 6 are excluded from the table due to poor performance.

Here and later when presenting computational results, "value" and "time" refer to objective function value and running time of an algorithm. The best solution value among all tested heuristics is shown in bold font. We also report (averaged over 10 instances of given type and size) the average solution value $\mathcal{A}(Q, C, D)$ (denoted simply as $\mathcal{A}$), computed using the closed-form expression from Section 2.3.3.

Table 3.1: Solution value and running time in seconds for construction heuristics

		RandomXYGreedy		Greedy		GreedyRandomized2		GreedyRandomized4		Rounding	
instances	$\mathcal{A}$	value	time	value	time	value	time	value	time	value	time
uniform 20x20	79975	62981	0.0011	61930	0.0016	61824	0.0015	62997	0.0023	**58587**	0.0282
uniform 40x40	1280013	1039365	0.0024	1038410	0.0085	1046862	0.0117	1047444	0.0107	**1005375**	0.4083
uniform 60x60	6480224	5335157	0.0057	5399004	0.0362	5430190	0.0403	5429077	0.0381	**5311287**	2.076
uniform 80x80	20480398	17179410	0.0119	17393975	0.0901	17427649	0.1092	17455112	0.1231	**17127745**	8.6041
uniform 100x100	50001181	**42492213**	0.0205	43134618	0.1797	43115743	0.1755	43209207	0.2431	42521606	29.3038
uniform 120x120	103680291	**88710617**	0.0334	90317432	0.2459	90450040	0.3127	90388890	0.3208	89342939	90.1245
uniform 140x140	192079012	**165656443**	0.0518	168664018	0.404	168695610	0.5922	168683177	0.5869	166927409	196.3766
uniform 160x160	327679690	**284623314**	0.0768	289819325	0.939	289847112	0.9922	290034508	0.9862	287148038	339.6329
uniform 180x180	524879096	**458395075**	0.1088	466419210	1.0135	466652862	1.107	466938203	1.5316	462852252	539.6931
normal 20x20	79977	69989	0.0011	69032	0.0013	69322	0.0015	69899	0.0022	**67367**	0.0275
normal 40x40	1280007	1137550	0.0022	1137478	0.008	1139150	0.0098	1139608	0.0116	**1123670**	0.3902
normal 60x60	6480142	5825775	0.0055	5847641	0.0229	5841178	0.0277	5860741	0.0427	**5795676**	2.0257
normal 80x80	20480028	18555962	0.0108	18696934	0.0613	18658585	0.0772	18697475	0.102	**18544051**	6.9208
normal 100x100	50000062	45647505	0.02	45909621	0.1293	45925799	0.1584	45943220	0.1958	**45643447**	30.2969
normal 120x120	103680643	**94952757**	0.0325	95765991	0.2465	95711199	0.2967	95757531	0.3385	95332171	80.9744
normal 140x140	192079732	**176656351**	0.0507	178279212	0.4034	178238835	0.4936	178233293	0.556	177501940	179.0639
normal 160x160	327681533	**302496650**	0.0738	305379404	0.746	305333912	0.696	305345983	0.823	304080792	310.9162
normal 180x180	524880349	**486132477**	0.1056	490345723	0.8888	490464093	1.0742	490656416	1.3211	489077716	540.4644
euclidean 20x20	95297	93756	0.0011	98864	0.0013	99027	0.0014	98104	0.0015	**85564**	0.0276
euclidean 40x40	1554313	1540492	0.0024	1559829	0.0111	1546894	0.0116	1551881	0.0123	**1430068**	0.4218
euclidean 60x60	8003105	7821082	0.0063	8021089	0.0445	8014594	0.0461	7945751	0.0489	**7331236**	1.9805
euclidean 80x80	24906273	24190227	0.0129	24873255	0.0611	24799662	0.0954	24853670	0.0805	**23145446**	6.141
euclidean 100x100	61053265	59345477	0.0235	60305521	0.103	59882626	0.1285	60052837	0.1223	**56848260**	31.8484
euclidean 120x120	126198999	121816738	0.0389	123601338	0.2986	123829252	0.305	124053452	0.3252	**117754675**	93.6024
euclidean 140x140	230673448	221785417	0.0617	227949036	0.4082	227508295	0.4637	227854403	0.4979	**214876628**	183.0906
euclidean 160x160	404912898	390412111	0.0897	395260253	0.8908	398388924	0.8284	396277525	1.0551	**378608021**	309.2262
euclidean 180x180	635700756	607470603	0.1289	623035384	1.1913	625456121	1.356	623393649	1.4349	**593800828**	548.8153

As the table shows, for smaller *uniform* and *normal* instances as well as for all *euclidean* instances *Rounding* produced better quality results, however, using substantially longer time. For all other problems *RandomXYGreedy* obtained better results. To our surprise, the quality of the solution produced by *Greedy* was inferior to that of *RandomXYGreedy*. It can, perhaps, be explained as a consequence of being "too greedy" in the beginning, leading to worse overall solution, particularly, taking into consideration the quadratic nature of the objective function. In the initial steps the choice is made based on the very much incomplete information about solution and the interaction cost of $\mathbf{x}$ and $\mathbf{y}$ assignments. In addition, the running time for *RandomXYGreedy* was significantly lower than that of *Rounding* and other algorithms. Thus, we conclude that *RandomXYGreedy* is our method of choice if a solution to BAP is needed quickly.

As for the *GreedyRandomized* strategy, the higher the size of the candidate list, the worse is the quality of the resulting solution. On the other hand, larger sizes of the candidate lists provide us with more diversified ways to generate solutions for BAP. That may have advantages if the construction is followed by an improvement approach as generally done in GRASP algorithm.

In Figures 3.2 and 3.3 we present solution value and running time results of this section for *uniform* instances.

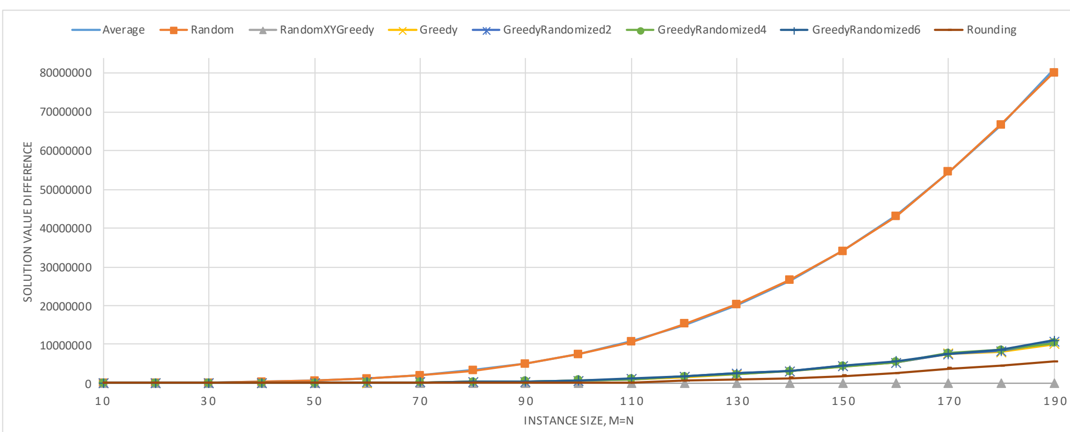

Figure 3.2: Difference between solution values (to the best) for construction heuristics; *uniform* instances

3.3.3 Experimental analysis of local search algorithms

Let us now discuss the results of computational experiments carried out using local search algorithms that explore neighborhoods discussed in Section 3.2. All algorithms are started from the same random solution and ran until a local optimum is reached. In addition to

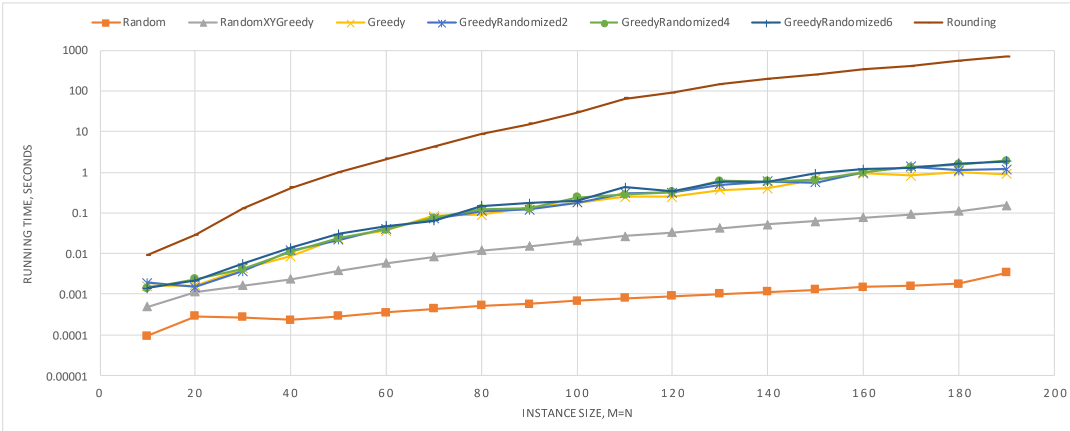

Figure 3.3: Running time for construction heuristics; *uniform* instances

the objective function value and running time we report the number of iterations for each approach.

For h-exchange neighborhoods, we selected 2 and 3-exchange local search algorithms (denoted by ***2ex*** and ***3ex***) as well as the Alternating Algorithm (***AA***).

From $[h, p]$-exchange based algorithms, we have implemented $[2, 2]$-exchange local search (named ***Dual2ex***). The $[2, 2]$-exchange neighborhood can be explored in $O(m^2n^2)$ time, using efficient recomputation of the change in the objective value. We refer to the algorithm that explores optimized 2-exchange neighborhood as ***2exOpt***. The running time of each iteration of this local search is $O(m^2n^3)$. To speed up this potentially slow approach, we have also considered a version, namely ***2exOptHeuristic***, where we use an $O(n^2)$ heuristic to solve the underlying linear assignment problem, instead of the Hungarian algorithm with cubic running time. The running time of each iteration of 2exOptHeuristic is then $O(m^2n^2)$. Similarly defined will be ***3exOpt***.

Shift, ***ShiftShuffle***, ***DualShift*** and ***ShiftOpt*** are implementations of local search based on shift, shift+shuffle, dual shift and optimized shift neighborhoods respectively.

In addition, we consider variations of the above-mentioned algorithms, namely ***2exFirst***, ***3exFirst***, ***Dual2exFirst***, ***2exOptFirst***, ***2exOptHeuristicFirst***, ***ShiftOptFirst***, where corresponding neighborhoods explored only until the first improving solution is encountered.

We provide a summary of complexity results on these local search algorithms in Table 3.2. Here by I we denote the number of iterations (or "moves") that it takes for a corresponding search to converge to a local optimum. As I could potentially be exponential in n and will vary between algorithms, we use this notation to simply emphasize the running time of an iteration of each approach.

Table 3.2: Asymptotic running time and neighborhood size per iteration for local searches

name	running time	neighborhood size per iteration
2ex	$O(n^3 + In^2)$	$\Theta(n^2)$
Shift	$O(In^3)$	n
ShiftShuffle	$O(In^3)$	$\Theta(n^2)$
3ex	$O(In^3)$	$\Theta(n^3)$
AA	$O(In^3)$	$n!$
DualShift	$O(n^4)$	n^2
Dual2ex	$O(In^4)$	$\Theta(n^4)$
ShiftOpt	$O(In^4)$	$n \cdot n!$
2exOptHeuristic	$O(In^4)$	$\Theta(n^2 \cdot n!)^*$
2exOpt	$O(In^5)$	$\Theta(n^2 \cdot n!)$
3exOpt	$O(In^6)$	$\Theta(n^3 \cdot n!)$

* 2exOptHeuristic does not fully explore the neighborhood.

Table 3.3 summarizes experimental results for *2ex*, *3ex*, *AA*, *2exOpt* and *2exOptFirst*. Results for other algorithms are not included in the table due to inferior performance. However, figures 3.4 and 3.5 provide additional insight into the performance of all the algorithms we have tested, for the case of *uniform* instances.

Table 3.3: Solution value, running time in seconds and number of iterations for local searches

		2ex			3ex			AA			2exOpt			2exOptFirst		
instances	$\mathcal{A}$	value	time	iter	value	time	iter	value	time	iter	value	time	iter	value	time	iter
uniform 10x10	4995	3378	0.0	9	3241	0.0	9	3385	0.0	3	**3103**	0.04	4	3128	0.02	11
uniform 20x20	80043	59371	0.0	20	56593	0.01	18	56097	0.01	4	**54912**	0.68	6	55059	0.34	25
uniform 30x30	404944	310455	0.02	32	297569	0.05	28	298787	0.02	4	291520	3.96	6	**291268**	3.09	46
uniform 40x40	1279785	1003731	0.04	45	977498	0.14	39	971400	0.06	5	**954676**	21.71	10	957381	8.46	56
uniform 50x50	3124809	2493822	0.08	57	2433665	0.32	49	2416832	0.13	5	**2385232**	63.49	11	2389496	24.94	73
uniform 60x60	6479878	5256357	0.15	74	5149634	0.59	55	5098653	0.26	6	5056566	143.48	11	**5031368**	80.32	97
uniform 70x70	12005619	9844646	0.24	85	9682798	1.1	67	9587489	0.38	6	**9469736**	326.04	14	9472549	156.78	114
uniform 80x80	20480209	17022523	0.37	96	16694088	1.81	75	16519908	0.66	7	16388545	504.34	12	**16355658**	285.23	136
uniform 90x90	32803918	27479017	0.52	111	26978715	2.97	88	26650508	1.08	8	26563051	882.81	13	**26514860**	497.74	158
uniform 100x100	49999078	42138227	0.74	124	41363121	4.96	109	41031842	1.45	8	40912367	1480.03	14	**40767754**	864.39	172
uniform 110x110	73206906	61988038	1.06	148	61179121	6.57	109	60529975	1.92	7	60162728	2406.29	15	**60068824**	1504.27	196
uniform 120x120	103679901	88602187	1.23	137	87330165	8.52	109	86174642	2.61	8	85872203	3865.67	18	**85670906**	1917.76	201
normal 10x10	4999	4044	0.0	10	4019	0.0	9	4040	0.0	2	3910	0.03	4	**3862**	0.02	14
normal 20x20	79955	67321	0.0	20	66520	0.01	16	66179	0.01	3	**64913**	0.79	7	65363	0.33	25
normal 30x30	404959	348058	0.02	34	342238	0.06	29	343639	0.03	4	**338796**	4.98	8	339162	2.61	45
normal 40x40	1279974	1119684	0.04	46	1111127	0.14	33	1099106	0.07	6	1089996	23.21	10	**1089752**	10.61	60
normal 50x50	3124879	2752326	0.08	63	2737137	0.34	43	2711191	0.14	6	2696287	65.48	11	**2696062**	32.57	77
normal 60x60	6479794	5769522	0.16	73	5707107	0.7	53	5665027	0.3	7	5640412	151.84	12	**5633463**	81.97	99
normal 70x70	12004939	10738678	0.24	88	10641129	1.3	65	10596245	0.42	6	10544640	316.24	13	**10538513**	144.42	116
normal 80x80	20480106	18434378	0.38	103	18282395	2.35	80	18173927	0.71	7	18126933	537.29	12	**18095224**	338.76	132
normal 90x90	32805972	29736595	0.51	108	29408513	3.79	91	29245481	0.92	6	29176212	1017.08	14	**29165974**	500.62	151
normal 100x100	49999105	45514117	0.71	122	45009249	5.69	100	44798388	1.45	7	44635991	1602.09	15	**44603238**	940.19	176
normal 110x110	73205050	66768499	1.01	142	66224593	8.26	110	65812495	2.69	10	65716978	2218.71	13	**65539744**	1632.32	193
normal 120x120	103681336	95001950	1.32	147	94151507	11.24	116	93702171	2.16	6	93322807	4645.28	20	**93248160**	2130.64	215
euclidean 10x10	6186	5397	0.0	13	5379	0.0	12	5404	0.0	3	**5368**	0.05	4	5375	0.03	16
euclidean 20x20	95834	82325	0.01	41	82293	0.01	25	82242	0.01	3	82160	1.27	5	**81813**	1.52	49
euclidean 30x30	490614	419174	0.02	61	418942	0.07	40	419000	0.03	3	**416436**	9.13	5	417339	18.19	98
euclidean 40x40	1553544	1314659	0.07	87	1312649	0.21	59	1311131	0.07	3	**1309701**	37.08	5	1311093	90.91	156
euclidean 50x50	3761359	3178424	0.14	112	3173915	0.5	78	3178006	0.16	4	**3167772**	134.77	7	3168388	314.91	211
euclidean 60x60	7999029	6740779	0.26	141	6720560	1.04	98	**6714400**	0.23	4	6714689	314.14	7	6716877	1012.93	296
euclidean 70x70	14909550	12533959	0.45	180	12500249	1.92	117	12490034	0.42	4	**12487021**	674.66	7	12499281	2354.68	366
euclidean 80x80	25210773	21188706	0.68	200	21182227	3.2	133	21160309	0.55	4	**21150070**	1222.19	6	21156445	5250.01	456
euclidean 90x90	39495474	33083033	1.04	240	33072079	4.87	145	33082326	0.98	5	**33049474**	2017.96	6	33089283	10482.75	556

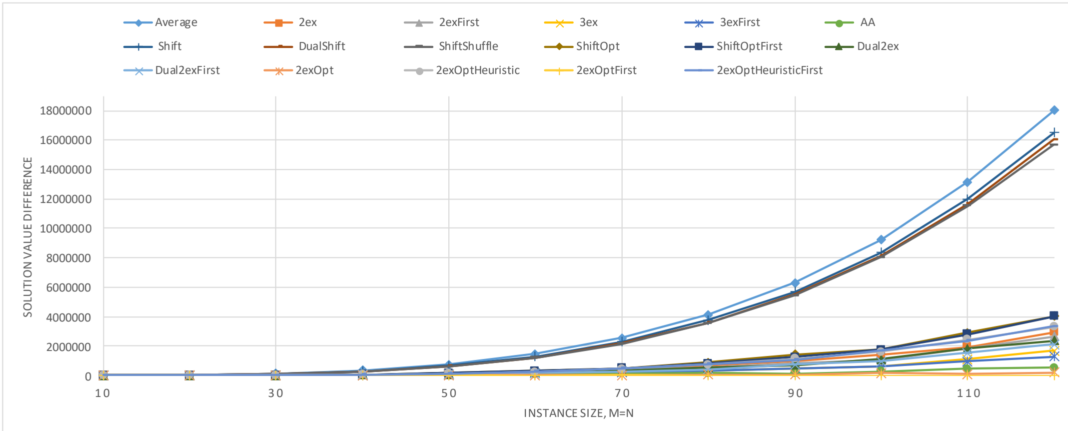

Figure 3.4: Difference between solution values (to the best) for local search; *uniform* instances

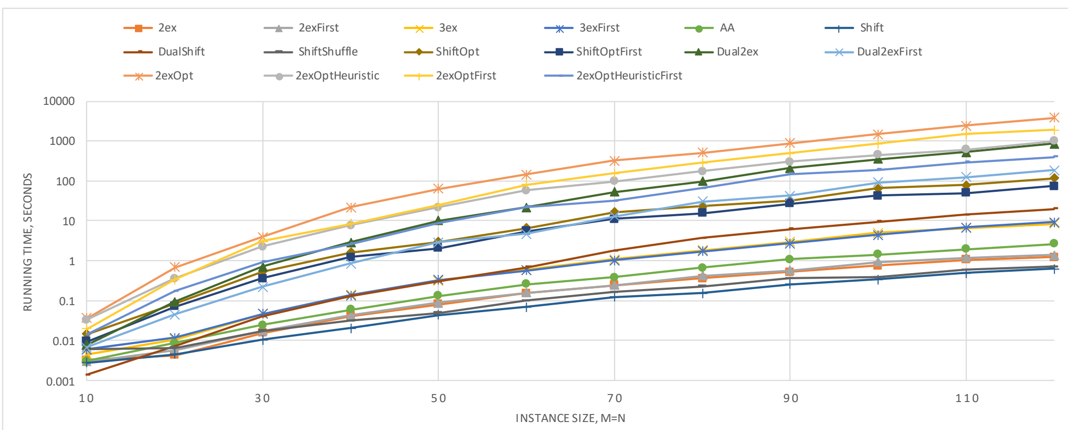

Figure 3.5: Running time to converge for local search; *uniform* instances

Even though the convergence speed is very fast for implementations of *Shift*, *ShiftShuffle* and *DualShift*, the resulting solution values are not significantly better than the average value $\mathcal{A}(Q, C, D)$ for the instance.

The *optimized shift* versions, namely *ShiftOpt* and *ShiftOptFirst* produced better solutions but still are outperformed by all remaining heuristics. This fact together with the slower convergence speed (as compared to say *2ex*) shows the weaknesses of the approach.

Dual2ex and *Dual2exFirst* are heavily outperformed both in terms of convergence speed as well as the quality of the resulting solution by *AA*.

It is also worth mentioning that speeding up *2exOpt* and *2exOptFirst* by substituting the Hungarian algorithm with an $O(n^2)$ heuristic for the assignment problem did not provide us with good results. The solution quality decreased substantially and, considering that the running time to converge is still slower than that of AA, we discard these options.

Table 3.3 presents the results for the better performing set of algorithms. The performance of both *first improvement* and *best improvement* approaches *2exFirst*, *3exFirst* and *2ex*, *3ex* respectively are similar so we will consider only the latter two from now on. Interestingly, it is not the case for the *optimized* neighborhoods. We noticed that, for *uniform* and *normal* instances *2exOptFirst* runs faster than *2exOpt*, in most cases. However, for *euclidean* instances *2exOptFirst* takes more time to converge.

As expected, AA is better than *3ex* with respect to both solution quality and running time. We will not include any of the h-exchange neighborhood search implementations for $h > 3$ in this study due to relatively poor performance and huge running time.

We focused the remaining experiments in the paper on *2ex*, AA and *2exOpt*. Among these *2ex* converges the fastest, *2exOpt* provides the best solutions and AA assumes a "balanced" position. It is also clear that even better solution quality could be achieved by using implementations of optimized h-exchange neighborhood search with higher h. However, we show in the next sub-section that this is not feasible in terms of efficient metaheuristics implementation.

Local search with multi-start

Now we would like to see how well our heuristics perform in terms of solutions quality, when the amount of time is fixed. For this we implemented a simple multi-start strategy for each of the algorithms. The framework will keep restarting the local search from the new *Random* instance until the time limit is reached. The best solution found in the process is then reported as the result.

Time limit for each instance will be set as the following. Considering the results of the previous sub-section, we expect *3exOptFirst* to be the slowest method to converge for all of the instances. We run it exactly once, and use its running time as a time limit for other multi-start algorithms. Together with resulting values we also report the number of restarts of each approach in Table 3.4. Clearly, the choice of time limit yields 1 as the number of starts for *3exOptFirst*.

Table 3.4: Solution value and number of starts for time-limited multi-start local searches

		3exOptFirst		2exOpt		2exOptFirst		AA		2ex		2exFirst	
instances	time limit	value	starts	value	starts	value	starts	value	starts	value	starts	value	starts
uniform 10x10	0.1	3059	1	2943	3	2974	5	2946	97	**2934**	221	2980	176
uniform 20x20	2.7	54250	1	53496	4	53286	8	**53096**	428	53983	997	54244	879
uniform 30x30	23.4	290200	1	288401	5	285630	10	**285271**	919	292991	1859	292363	1695
uniform 40x40	103.2	948029	1	943982	5	940718	10	**936113**	1528	963120	2858	960093	2679
uniform 50x50	531.7	2370639	1	2365473	8	2358811	18	**2346865**	3664	2410678	6592	2401247	6337
uniform 60x60	1148.5	5017422	1	5003247	7	4989212	16	**4980930**	4221	5105064	7747	5092544	7522
uniform 70x70	3291.3	9429464	1	9421085	10	9404126	21	**9369944**	7017	9601891	13499	9583585	13009
uniform 80x80	3763.3	16406602	1	16319588	7	16241213	13	**16229861**	5031	16612105	10017	16583987	9578
normal 10x10	0.1	3857	1	3838	2	3851	5	3828	91	**3818**	208	3847	162
normal 20x20	2.5	65014	1	64635	4	64433	7	**64020**	396	64867	902	64738	769
normal 30x30	23.4	337626	1	336552	5	335378	10	**335042**	899	339448	1818	338849	1623
normal 40x40	113.3	1086083	1	1082094	5	1081530	12	**1078755**	1675	1092923	3063	1091803	2840
normal 50x50	469.3	2688595	1	2679334	8	2677720	16	**2672481**	3217	2711913	5807	2704948	5475
normal 60x60	933.4	5640721	1	5627391	6	5612362	13	**5604229**	3413	5679037	6216	5672749	5979
normal 70x70	3593.3	10512493	1	10492591	12	10483432	25	**10474343**	7685	10604646	14559	10591133	13903
normal 80x80	11339.0	**17989971**	1	17993643	20	18010732	42	17995894	15435	18226724	29827	18209532	28425
euclidean 10x10	0.1	5447	1	5430	3	5445	3	**5427**	98	5427	266	5427	162
euclidean 20x20	5.1	82409	1	81717	4	81710	4	**81573**	589	81573	1283	81575	747
euclidean 30x30	70.1	418658	1	415529	7	415419	4	**414767**	2399	414774	3382	414808	1732
euclidean 40x40	390.3	1321385	1	1317439	9	1317948	4	**1316409**	5459	1316509	6197	1316771	3010
euclidean 50x50	1675.4	3151591	1	3136628	13	3139866	4	**3135362**	11411	3135723	11993	3136122	5359
euclidean 60x60	4604.9	6563921	1	6532789	15	6537657	4	**6529495**	17621	6530835	17448	6532247	6641

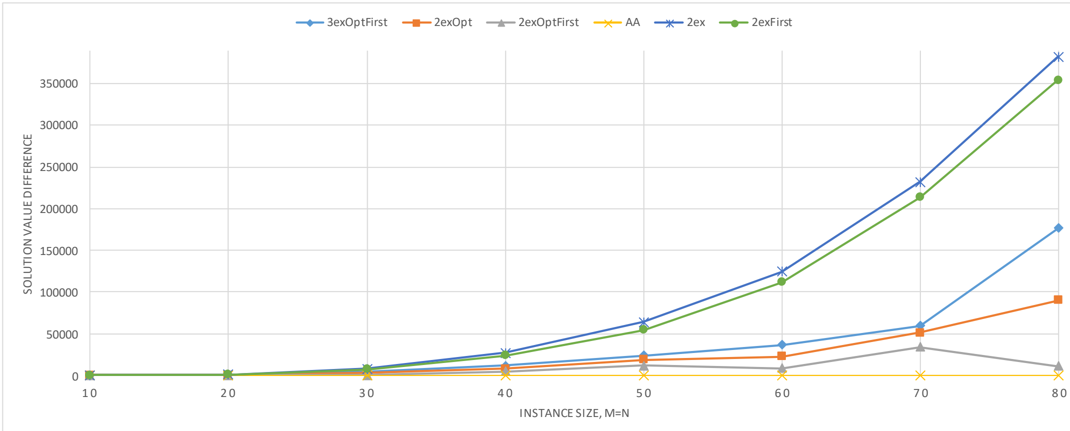

Figure 3.6: Difference between solution values (to the best) for multi-start algorithms; *uniform* instances

The best algorithm in these settings is *AA*, which consistently exhibited better performance for all instance types. The reason behind this is the fact that a local optimum by this approach can be reached almost as fast as by *2ex*, however solution quality is much better. On the other hand, the convergence of *2exOpt* to a local optimum is very time consuming, and perhaps a better strategy is to do more restarts with slightly less quality of resulting solution. Similar argument holds for the case why *2exOptFirst* outperforms *3exOptFirst* in this type of experiments. This observation is in contrast with the results experienced by researches of bipartite unconstrained binary quadratic program [54] and bipartite quadratic assignment problem [109]. The difference can be attributed to the more complex structure of BAP in comparison to problems mentioned above.

3.3.4 Variable neighborhood search

Variable neighborhood search (VNS) is an algorithmic paradigm to enhance standard local search by making use of properties (often complementary) of multiple neighborhoods [3, 69]. The 2-exchange neighborhood is very fast to explore and optimized 2-exchange is more powerful but searching through it for an improving solution takes significantly more time. The neighborhood considered in the *Alternating Algorithm* works better when significant asymmetry is present regarding $\mathbf{x}$ and $\mathbf{y}$ variables. Motivated by these complementary properties, we have explored VNS based algorithms to solve BAP.

We start by attempting to improve the convergence speed of *AA* by the means of the faster *2ex*. The first variation, named ***2ex+AA*** will first apply *2ex* to *Random* starting solution and then apply *AA* to the resulting solution. A more complex approach ***2exAAStep*** (Algorithm 4) will start by applying *2ex* and as soon as the search converge it will apply a single improvement (step) with respect to *Alternating Algorithm* neighborhood. After successful update the procedure defaults to running *2ex* again. The process stops when no more improvements by AA (and consequently by *2ex*) are possible.

Algorithm 4 *2exAAStep*

Input: integers m, n; $m \times m \times n \times n$ array Q; feasible solution $(\mathbf{x}, \mathbf{y})$ to given BAP
Output: feasible solution to given BAP
  while True **do**
    $(\mathbf{x}, \mathbf{y}) \leftarrow 2ex(m, n, Q, (\mathbf{x}, \mathbf{y}))$ ▷ running 2-exchange local search (Section 3.2.1)
    $e_{ij} \leftarrow \sum_{k,l \in N} q_{ijkl} y_{kl} \; \forall i, j \in M$
    $\mathbf{x}^* \leftarrow arg\,min_{\mathbf{x}' \in \mathcal{X}} \sum_{i,j \in M} e_{ij} x'_{ij}$ ▷ solving assignment problem for **x**
    if $f(\mathbf{x}^*, \mathbf{y}) < f(\mathbf{x}, \mathbf{y})$ **then**
      continue ▷ restarting the procedure **while** loop
    end if
    $g_{kl} \leftarrow \sum_{i,j \in M} q_{ijkl} x^*_{ij} \; \forall k, l \in N$
    $\mathbf{y}^* \leftarrow arg\,min_{\mathbf{y}' \in \mathcal{Y}} \sum_{k,l \in N} g_{kl} y'_{kl}$ ▷ solving assignment problem for **y**
    if $f(\mathbf{x}^*, \mathbf{y}^*) = f(\mathbf{x}, \mathbf{y})$ **then**
      break ▷ algorithm converged, terminate
    end if
    $\mathbf{x} \leftarrow \mathbf{x}^*$; $\mathbf{y} \leftarrow \mathbf{y}^*$
  end while
  return $(\mathbf{x}, \mathbf{y})$

Results in Table 3.5 follow the structure of experimental results reported earlier in the paper. The number of iterations that we report for *2exAAStep* is the number of times the heuristic switches from 2-exchange neighborhood to the neighborhood of the *Alternating Algorithm*. Clearly, this number will be 1 for *2ex+AA* by design.

As all these approaches are guaranteed to be locally optimal with respect to *Alternating Algorithm* neighborhood, we expect the solution values to be similar. This can be seen in the table. A main observation here is that the *2ex* heuristic does not combine well with AA. Increased running time for both *2ex+AA* and *2exAAStep* confirms that AA is more efficient in searching its much larger neighborhood.

We then explored the effect of combining *2exOptFirst* and AA. An algorithm that first runs AA once and then applies *2exOptFirst* until convergence will be referred to as ***AA+2exOptFirst***. A more desirable variable neighborhood search based on the discussed heuristics will use the fact that most of the time running AA until convergence is faster than even a single update of the solutions during the *2exOptFirst* run. The algorithm ***AA2exOptFirstStep*** (Algorithm 5) will use AA to reach its local optimum and then will try to escape it by applying a single first possible improvement of the slower search *2exOptFirst*. If successful, the process will start from the beginning with AA. We will also add to the comparison variation with *best improvement rule*, namely ***AA2exOptStep***.

The results of these experiments are reported in Table 3.6. Here, we also report the number of iterations for *AA2exOptStep* and *AA2exOptFirstStep*, which represents the number of switches from the *Alternating Algorithm* neighborhood to optimized 2-exchange neighborhood before the algorithms converge.

Algorithm 5 *AA2exOptFirstStep*

Input: integers m, n; $m \times m \times n \times n$ array Q; feasible solution $(\mathbf{x}, \mathbf{y})$ to given BAP
Output: feasible solution to given BAP

  while True **do**
    $(\mathbf{x}, \mathbf{y}) \leftarrow AA(m, n, Q, (\mathbf{x}, \mathbf{y}))$ ▷ running *Alternating Algorithm* (Section 3.2.1)
    for all $i_1 \in M$ **and all** $i_2 \in M \setminus \{i_1\}$ **do**
      $j_1 \leftarrow$ assigned index to i_1 in $\mathbf{x}$
      $j_2 \leftarrow$ assigned index to i_2 in $\mathbf{x}$

      $\mathbf{x}^* \leftarrow \mathbf{x}$
      $x^*_{i_1 j_1} \leftarrow 0$; $x^*_{i_2 j_2} \leftarrow 0$; $x^*_{i_1 j_2} \leftarrow 1$; $x^*_{i_2 j_1} \leftarrow 1$ ▷ applying 2-exchange
      $g_{kl} \leftarrow \sum_{i,j \in M} q_{ijkl} x^*_{ij}\ \forall k, l \in N$
      $\mathbf{y}^* \leftarrow arg\,min_{\mathbf{y}' \in \mathcal{Y}} \sum_{k,l \in N} g_{kl} y'_{kl}$ ▷ solving assignment problem for $\mathbf{y}$
      if $f(\mathbf{x}^*, \mathbf{y}^*) < f(\mathbf{x}, \mathbf{y})$ **then**
        $\mathbf{x} \leftarrow \mathbf{x}^*$; $\mathbf{y} \leftarrow \mathbf{y}^*$
        continue while ▷ restarting the procedure **while** loop
      end if
    end for
    for all $k_1 \in N$ **and all** $k_2 \in N \setminus \{k_1\}$ **do**
      $l_1 \leftarrow$ assigned index to k_1 in $\mathbf{y}$
      $l_2 \leftarrow$ assigned index to k_2 in $\mathbf{y}$

      $\mathbf{y}^* \leftarrow \mathbf{y}$
      $y^*_{k_1 l_1} \leftarrow 0$; $y^*_{k_2 l_2} \leftarrow 0$; $y^*_{k_1 l_2} \leftarrow 1$; $y^*_{k_2 l_1} \leftarrow 1$ ▷ applying 2-exchange
      $e_{ij} \leftarrow \sum_{k,l \in N} q_{ijkl} y^*_{kl}\ \forall i, j \in M$
      $\mathbf{x}^* \leftarrow arg\,min_{\mathbf{x}' \in \mathcal{X}} \sum_{i,j \in M} e_{ij} x'_{ij}$ ▷ solving assignment problem for $\mathbf{x}$
      if $f(\mathbf{x}^*, \mathbf{y}^*) < f(\mathbf{x}, \mathbf{y})$ **then**
        $\mathbf{x} \leftarrow \mathbf{x}^*$; $\mathbf{y} \leftarrow \mathbf{y}^*$
        continue while ▷ restarting the procedure **while** loop
      end if
    end for
      break ▷ algorithm converged, terminate
  end while
  return $(\mathbf{x}, \mathbf{y})$

Table 3.5: Solution value, running time in seconds and number of iterations for *Alternating Algorithm* and variations (convergence to local optima)

	AA		2ex+AA		2exAAStep		
instances	value	time	value	time	value	time	iter
uniform 10x10	3255	**0.0**	3305	0.0	3322	0.01	1
uniform 20x20	56287	**0.01**	56136	0.01	56076	0.01	3
uniform 30x30	297819	**0.02**	298485	0.03	297874	0.05	4
uniform 40x40	965875	**0.06**	967373	0.08	971010	0.13	5
uniform 50x50	2415720	**0.11**	2414279	0.18	2419385	0.34	6
uniform 60x60	5077348	**0.23**	5089275	0.33	5095460	0.77	9
uniform 70x70	9578626	**0.32**	9561747	0.51	9549687	1.25	10
uniform 80x80	16505833	**0.59**	16422705	0.93	16474525	1.87	10
uniform 90x90	26650437	**0.93**	26726070	1.16	26706156	3.04	11
uniform 100x100	41027445	**1.12**	41001387	1.89	41038180	4.78	14
uniform 110x110	60512662	**1.72**	60549540	2.37	60508210	6.87	15
uniform 120x120	86397256	**2.08**	86108044	3.23	86019130	10.47	18
uniform 130x130	119380881	**3.02**	119421396	4.06	119417016	12.52	16
uniform 140x140	161524589	**3.58**	161725915	5.6	161535754	16.97	18
uniform 150x150	213377462	**5.02**	214064556	6.9	213453225	22.48	19
normal 10x10	4037	**0.0**	3997	0.0	3997	0.0	2
normal 20x20	66006	**0.01**	66372	0.01	66104	0.01	3
normal 30x30	343319	**0.02**	342316	0.03	342776	0.05	3
normal 40x40	1096961	**0.06**	1098741	0.09	1101256	0.17	7
normal 50x50	2712329	**0.12**	2709929	0.2	2708557	0.38	8
normal 60x60	5668986	**0.21**	5671907	0.33	5678451	0.72	8
normal 70x70	10561145	**0.42**	10588835	0.57	10581535	1.29	10
normal 80x80	18172093	**0.51**	18160338	0.87	18141092	2.22	12
normal 90x90	29222387	**0.91**	29231041	1.3	29283340	2.84	10
normal 100x100	44751122	**1.31**	44735031	1.72	44753417	5.22	15
normal 110x110	65809366	**1.64**	65817524	2.39	65812802	6.97	15
normal 120x120	93529513	**2.26**	93491028	3.58	93581308	8.65	14
normal 130x130	129150096	**3.26**	129310194	4.14	129238943	12.84	17
normal 140x140	174245361	**3.75**	174296950	5.91	174169032	20.14	21
normal 150x150	230484514	**4.28**	230242366	7.32	230292305	24.21	21
euclidean 10x10	5032	**0.0**	5015	0.0	5015	0.01	1
euclidean 20x20	81714	**0.01**	81701	0.01	81701	0.01	2
euclidean 30x30	424425	**0.03**	424261	0.04	424261	0.06	3
euclidean 40x40	1331726	**0.06**	1330070	0.11	1330070	0.15	4
euclidean 50x50	3342515	**0.13**	3337157	0.24	3337157	0.35	4
euclidean 60x60	6637101	**0.24**	6622844	0.42	6622844	0.63	5
euclidean 70x70	12373648	**0.33**	12345122	0.7	12345122	1.01	4
euclidean 80x80	21088451	**0.55**	21060424	1.01	21060424	1.34	3
euclidean 90x90	33842019	**0.85**	33831315	1.48	33831315	2.01	4
euclidean 100x100	50386904	**1.08**	50351081	2.19	50350547	3.33	5

Table 3.6: Solution value, running time in seconds and number of iterations for *2exOpt* and variations (convergence to local optima)

	2ExOptFirst		AA+2exOptFirst		AA2exOptStep			AA2exOptFirstStep		
instances	value	time	value	time	value	time	iter	value	time	iter
uniform 10x10	3156	**0.02**	3059	0.01	3054	0.02	2	3059	0.02	3
uniform 20x20	54670	**0.35**	54877	0.24	54718	0.32	3	54431	0.2	4
uniform 30x30	291902	2.27	294044	**1.12**	291184	2.64	4	290011	1.17	4
uniform 40x40	948344	9.78	958550	**3.29**	953938	5.61	3	958215	3.4	3
uniform 50x50	2379856	33.02	2399151	11.15	2392151	16.57	3	2395319	**8.08**	3
uniform 60x60	5044883	64.73	5026000	36.08	5030618	35.22	3	5026865	**20.45**	4
uniform 70x70	9479099	168.6	9511756	67.85	9521222	78.27	3	9501548	**28.81**	3
uniform 80x80	16418360	252.23	16400987	120.41	16390406	132.04	3	16381373	**56.49**	3
uniform 90x90	26507000	569.45	26499481	229.48	26536687	238.11	3	26546135	**113.07**	4
uniform 100x100	40753550	878.32	40894844	293.74	40949795	184.26	1	40875653	**155.41**	3
uniform 110x110	60079399	1539.8	60231687	458.67	60277301	487.4	3	60196421	**317.3**	4
uniform 120x120	85818278	2120.85	85774789	1090.37	85996522	526.83	2	86070239	**306.32**	2
uniform 130x130	118773110	3515.46	118967905	1105.4	119034719	827.06	2	119133276	**452.11**	3
uniform 140x140	160780185	4860.32	160956538	1304.17	161002007	1479.93	3	161113803	**764.84**	3
uniform 150x150	213525103	5514.74	213372569	**538.34**	213372569	748.16	1	213372569	553.21	1
normal 10x10	3866	**0.02**	3895	0.01	3886	0.02	2	3917	0.01	2
normal 20x20	65262	**0.3**	65137	0.28	65166	0.36	3	65258	0.21	4
normal 30x30	338569	2.9	340096	1.19	340240	1.52	2	340534	**0.86**	3
normal 40x40	1087006	10.28	1087569	6.04	1090323	6.93	3	1089412	**3.46**	4
normal 50x50	2695007	26.39	2697747	14.44	2697124	19.13	3	2696860	**7.45**	3
normal 60x60	5637608	71.64	5639469	34.18	5634802	45.69	4	5638741	**18.75**	3
normal 70x70	10538891	159.53	10527751	61.85	10524931	80.22	3	10532494	**33.81**	3
normal 80x80	18102861	292.68	18102161	145.45	18123379	148.5	4	18125319	**62.56**	3
normal 90x90	29162243	447.82	29167487	166.29	29176575	193.61	3	29167084	**102.4**	3
normal 100x100	44610176	953.0	44644532	272.9	44626268	376.46	4	44645246	**153.74**	3
normal 110x110	65589378	1404.23	65635027	561.99	65669769	423.52	3	65646106	**233.42**	3
normal 120x120	93315766	2071.35	93321138	697.7	93338052	692.75	3	93300933	**346.7**	3
normal 130x130	128872342	3329.54	129005518	630.07	128978046	784.53	2	129030228	**361.45**	2
normal 140x140	173877153	4669.47	174004558	1379.7	174104009	857.84	2	174117705	**565.83**	2
normal 150x150	229879808	6572.92	229985798	2161.19	230286566	1481.59	3	230254077	**757.09**	3
euclidean 10x10	4988	**0.04**	4995	0.02	4992	0.02	1	4996	0.02	1
euclidean 20x20	81833	1.46	81644	**0.33**	81644	0.31	1	81644	0.28	1
euclidean 30x30	424227	17.82	424425	**1.63**	424425	1.65	1	424425	1.64	1
euclidean 40x40	1330114	84.25	1331592	**7.63**	1331592	7.28	1	1331592	7.24	1
euclidean 50x50	3344106	347.38	3342208	22.61	3342208	20.49	1	3342208	**18.78**	1
euclidean 60x60	6628784	968.15	6637101	**43.81**	6637101	43.91	1	6637101	43.81	1
euclidean 70x70	12343342	2404.75	12373648	**90.12**	12373648	90.34	1	12373648	90.1	1
euclidean 80x80	21098260	5579.32	21088451	**174.46**	21088451	174.94	1	21088451	174.98	1
euclidean 90x90	33892498	11440.65	33841998	333.66	33841998	338.05	1	33841998	**326.42**	1
euclidean 100x100	50313528	19808.73	50386904	**514.2**	50386904	515.13	1	50386904	514.74	1

We have noticed that incorporating *Alternating Algorithm* into *optimized 2-exchange* yields a much better performance, bringing the convergence time down by at least an order of magnitude. Among variations, *AA2exOptFirstStep* is consistently faster for *uniform* and *normal* instances. However, for *euclidean* instances performance of all variable neighborhood search algorithms is similar. In fact, for euclidean instances of all sizes the average number of switches between neighborhoods is 1, which implies that there is no possible improvement from the optimized 2-exchange neighborhood after the Alternating Algorithm has converged. Thus, the special structure of instances must be always considered when developing metaheuristics for BAP.

Results on convergence time for all described algorithms from this sub-section, for *uniform* instances, are given in Figure 3.7.

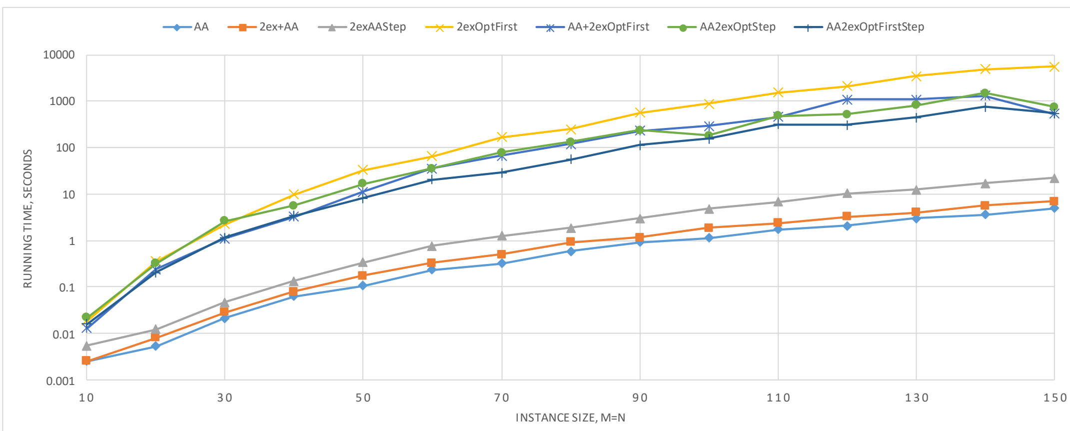

Figure 3.7: Running time to reach the local optima by algorithms; *uniform* instances

Our concluding set of experiments is dedicated to finding the most efficient combination of variable neighborhood search strategies and construction heuristics. We consider a variation of the VNS approach with the best convergence speed performance - *AA2exOptFirstStep*. Namely, let ***h-AA2exOptFirstStep*** be the algorithm that first generates h starting solution, using *RandomXYGreedy* strategy. It then proceeds to apply *AA* to each of these solutions, selecting the best one and discarding the rest. After that *h-AA2exOptFirstStep* will follow the description of *AA2exOptFirstStep* (Algorithm 5) and will alternate between finding an improving solution using optimized 2-exchange neighborhood and applying *AA*, until the convergence to local optima. In this sense, *AA2exOptFirstStep* and *1-AA2exOptFirstStep* are equivalent implementations.

The single iteration of *AA* requires $O(n^3)$ running time, whereas, a full exploration of the optimized 2-exchange neighborhood will take $O(m^2n^3)$. From the experiments in Section 3.3.3 we also know that it usually takes *AA* less than 10 iterations to converge.

Based on these observations, for the following experimental analysis we have chosen h for *h-AA2exOptFirstStep* as $h \in \{4, 10, 100\}$.

In addition to versions of *h-AA2exOptFirstStep* we consider a simple multi-start *AA* strategy that performed well in previous experiments (see Section 3.3.3), denoted ***msAA***. Now however, the starting solution each time is generated using *RandomXYGreedy* construction heuristic. As the time limit for this multi-start approach we select the highest convergence time among all *h-AA2exOptFirstStep* variations. As it often happens during the time-limited multi-start procedures, the best solution will be found before the final iteration. Hence, in addition to the total number we also report the average iteration (*best iter*) at which the finally reported solution was found, and the standard deviation of this value.

See the results of these experiments in Table 3.7 and Figure 3.8.

Table 3.7: Solution value, running time in seconds and number of iterations for Variable Neighborhood Search and multi-start AA

	AA2ExOptFirstStep			4AA2ExOptFirstStep			10AA2ExOptFirstStep			100AA2ExOptFirstStep			msAA				
instances	value	time	iter	value	time	iter	value	time	iter	value	time	iter	value	time	iter	best iter	σ(best iter)
uniform 10x10	3162	0.02	3	3126	0.01	1	3025	0.01	1	**2983**	0.06	1	2995	0.07	116	47	41
uniform 20x20	55131	0.15	2	54601	0.17	2	54294	0.19	2	**53281**	0.58	1	53620	0.59	131	54	37
uniform 30x30	293385	0.89	3	292039	0.83	2	289483	0.92	2	**286542**	2.42	1	287169	2.44	130	57	49
uniform 40x40	955295	3.03	3	950608	2.77	3	951947	2.87	2	942849	6.82	1	**939052**	6.85	138	89	32
uniform 50x50	2380817	11.35	5	2379835	11.88	4	2375551	7.42	2	2370805	15.35	1	**2360529**	16.56	165	83	52
uniform 60x60	5038934	19.96	3	5030082	15.28	2	5015756	18.16	2	**4990868**	35.36	2	4993774	38.42	208	112	35
uniform 70x70	9479825	34.21	4	9436974	43.32	3	9445502	39.85	3	9413893	54.29	1	**9399736**	61.76	203	115	67
uniform 80x80	16389632	61.47	3	16357168	55.61	2	16303348	59.12	2	**16261295**	95.21	1	16264848	104.0	217	95	54
uniform 90x90	26505894	110.55	3	26456700	94.5	3	26407075	80.08	1	26356116	151.23	2	**26342919**	160.45	226	83	64
uniform 100x100	40782492	141.59	3	40712949	180.44	3	40633567	165.63	3	40540438	208.3	1	**40506423**	241.3	241	116	96
uniform 120x120	85825930	342.18	3	85579139	274.87	2	85471530	333.39	3	85335239	441.49	1	**85283242**	509.31	273	122	71
uniform 140x140	160605657	693.67	3	160415349	555.54	3	160292924	474.41	1	160035009	719.05	1	**159912990**	927.88	286	131	124
uniform 160x160	277129402	909.79	2	276565751	918.66	2	276159588	908.23	2	**275721038**	1386.9	1	275725334	1657.71	302	154	100
normal 10x10	3894	0.02	3	3855	0.01	2	3855	0.01	1	**3808**	0.07	1	3809	0.07	117	40	37
normal 20x20	65712	0.15	2	65077	0.17	2	64803	0.2	1	**64293**	0.58	1	64477	0.58	130	73	48
normal 30x30	338547	1.17	5	337693	0.95	3	338138	0.79	1	**335113**	2.75	2	335756	2.76	145	74	42
normal 40x40	1090670	2.81	3	1088357	3.1	3	1085519	2.69	2	**1081375**	7.56	1	1082915	7.58	154	81	46
normal 50x50	2696368	8.24	3	2692035	8.33	2	2682121	8.66	3	**2678345**	17.52	2	2680271	17.58	175	71	55
normal 60x60	5647247	17.06	3	5633194	14.77	1	5627675	17.07	2	**5616899**	31.56	2	5617125	32.18	173	83	55
normal 70x70	10549768	26.89	1	10519922	34.7	3	10509205	30.19	2	**10493809**	57.37	2	10494503	61.86	201	104	64
normal 80x80	18095404	72.05	3	18069406	59.64	2	18067347	55.46	2	18032081	86.61	1	**18023497**	100.11	209	112	62
normal 90x90	29115217	107.77	3	29103538	103.37	2	29097191	95.29	2	29045978	165.73	2	**29027250**	187.3	264	120	71
normal 100x100	44618697	130.7	2	44578918	138.0	2	44556729	162.61	3	44484747	245.72	3	**44482231**	279.76	274	172	62
normal 120x120	93293438	343.2	3	93162243	313.92	2	93112300	309.4	2	93023046	506.08	2	**92984865**	540.0	282	149	93
normal 140x140	173820624	535.5	2	173653510	510.49	2	173594266	481.53	1	173434718	815.2	2	**173430869**	900.03	279	144	76
normal 160x160	298434202	967.33	2	297840806	899.65	2	297816150	1030.84	2	297540220	1211.89	1	**297480023**	1567.93	294	126	62
euclidean 10x10	5037	0.02	1	**5026**	0.02	1	5027	0.02	1	5026	0.11	1	5026	0.11	116	6	7
euclidean 20x20	82675	0.25	1	82008	0.26	1	81842	0.31	1	**81718**	1.0	1	81718	1.0	129	12	11
euclidean 30x30	411014	1.78	1	408739	1.72	1	407379	1.91	1	**406970**	4.23	1	406970	4.24	162	32	43
euclidean 40x40	1348302	6.68	1	1342159	6.99	1	1339683	7.09	1	1337792	12.69	1	**1337738**	12.72	204	48	58
euclidean 50x50	3231060	21.05	1	3219207	20.39	1	3214867	19.94	1	3210442	30.74	1	**3210280**	31.97	254	37	36
euclidean 60x60	6548901	44.42	1	6519075	44.82	1	6515800	46.24	1	6507833	65.26	1	**6507813**	65.41	304	32	23
euclidean 70x70	12315235	93.93	1	12283239	100.51	1	12264197	96.28	1	12257619	126.03	1	**12256435**	128.94	388	74	76
euclidean 80x80	21240164	187.89	1	21143316	183.3	1	21104571	185.35	1	21096255	229.53	1	**21095365**	232.0	459	144	132
euclidean 90x90	33385322	335.48	1	33323860	319.99	1	33296502	326.28	1	33279588	388.9	1	**33277417**	398.29	558	81	126
euclidean 100x100	51524424	530.7	1	51382552	535.98	1	51303227	538.1	1	51289100	632.49	1	**51286565**	633.16	597	158	133
euclidean 120x120	105192868	1291.27	1	105092433	1284.2	1	105037756	1404.01	1	104969850	1456.4	1	**104965462**	1556.45	908	93	112

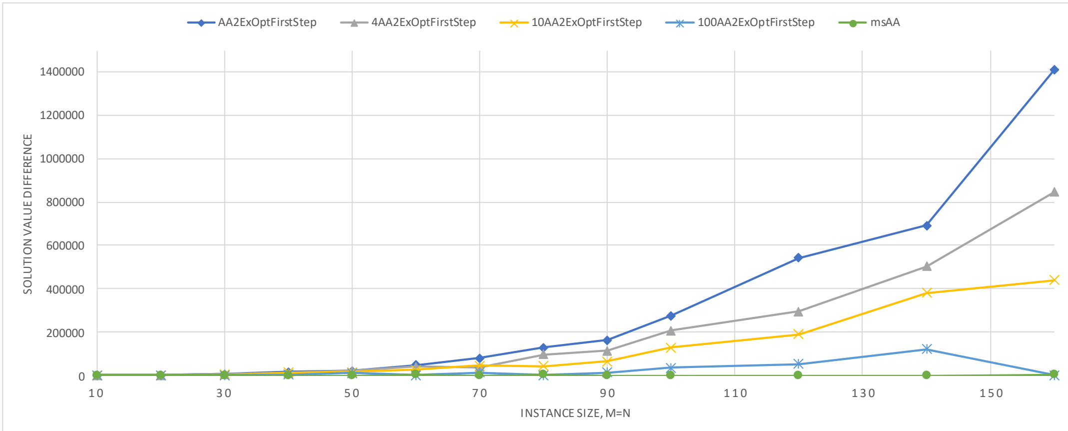

Figure 3.8: Difference between solution values (to the best) for algorithms; *uniform* instances

Under this considerations, multi-start AA once again performed the best. *h-AA2exOptFirstStep* variations were the more efficient, the higher the number h was. Interestingly, for several instance sizes, the average iteration of finding the best solution by *msAA* is substantially bellow 100. However, the observed standard deviation is very high, which hints towards the variability of the solutions produced by AA. To confirm this, we present in Figures 3.9, 3.10 and 3.11 the spread of solution values produced by applying AA to the solution of *RandomXYGreedy* (denoted as *RandomXYGreedy+AA*). All three instances in these charts are of size $m = n = 100$, and we perform 100 runs of this metaheuristic.

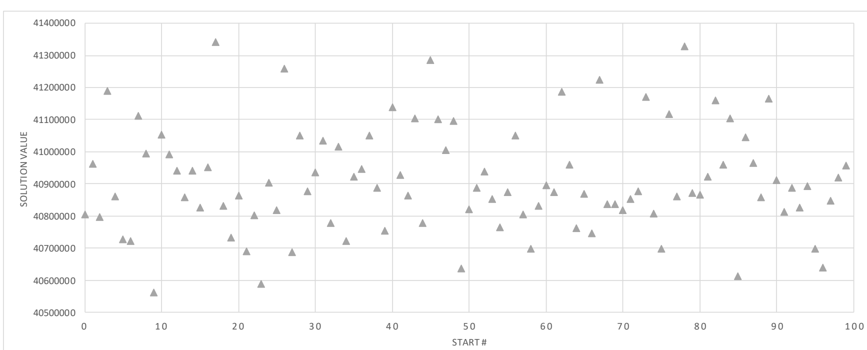

Figure 3.9: Objective solution values for *RandomXYGreedy+AA* metaheuristic; *uniform* 100×100 instance

At this point, we conclude that optimized 2-exchange neighborhood is too costly to explore, in comparison to the neighborhood that AA is based on. For the general case it is more effective to do several more restarts of AA from *RandomXYGreedy* solutions then to spend time escaping local optima with even a single step of *2exOpt*. It is suggested to

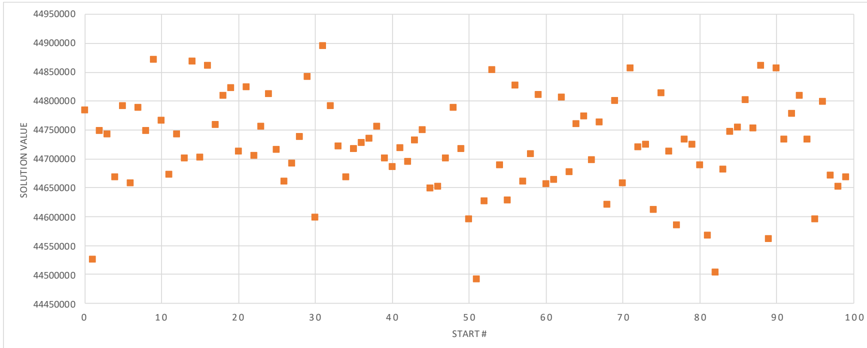

Figure 3.10: Objective solution values for *RandomXYGreedy+AA* metaheuristic; *normal* 100×100 instance

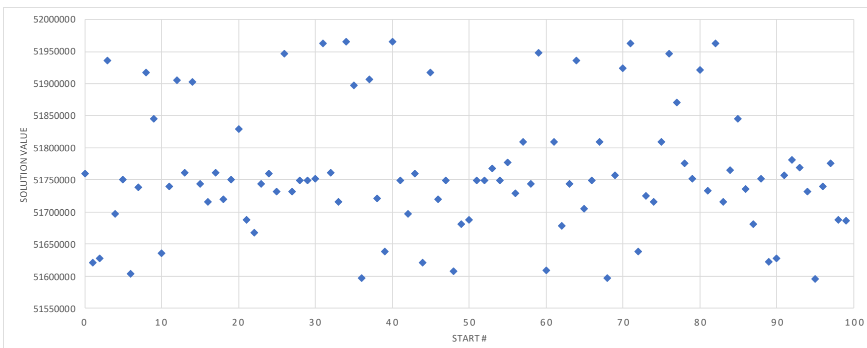

Figure 3.11: Objective solution values for *RandomXYGreedy+AA* metaheuristic; *euclidean* 100×100 instance

only use efficient implementations of VNS that explore optimized 2-exchange neighborhood as the final step of any metaheuristic. In this way you can improve your solution quality without excessive time spending, while leaving all the heavy work for *Alternation Algorithm*.

Our previous experiments that involve multi-start strategies (in this section and Section 3.3.3) have reasonable time limit restrictions. This considerations are important when developing algorithms to run on real-life instances. However, we are also interested in behavior of multi-start AA and multi-start VNS in the case of unlimited (or unreasonably large) running time constraints. Figure 3.12 presents results of running multi-start *AA*, multi-start *1-AA2exOptFirstStep* and multi-start *100-AA2exOptFirstStep*, for a single 100×100 *uniform* instance, for an exceedingly long period of time. All starts are made from the solutions generated by *RandomXYGreedy* heuristic. Here we report the change of the best found solution value, depending on time.

We can see that after 50000 seconds (0.58 days of running) multi-start VNS strategies begin to dominate the multi-start *AA*, even though the later approach is much more efficient

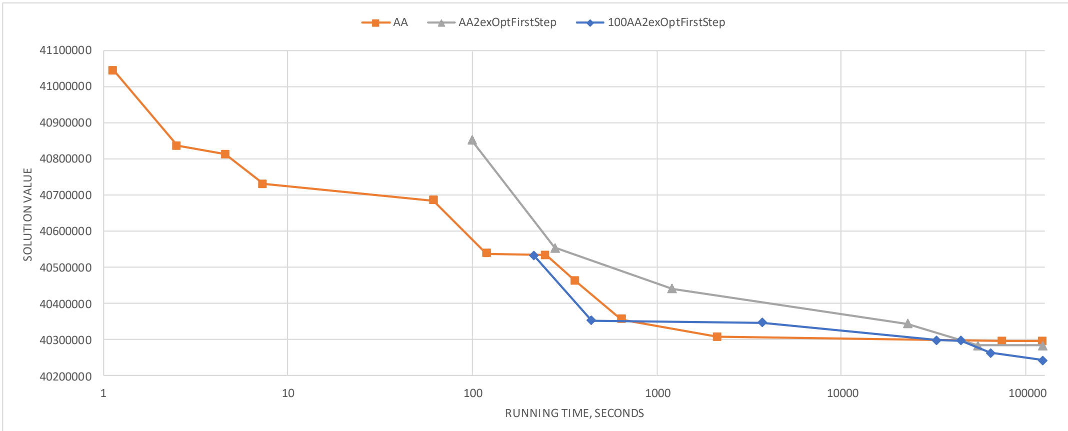

Figure 3.12: Improvement over time of best found objective solution value for multi-start heuristics; *uniform* 100×100 instance

in solution space exploration for short running times. This observation is consistent with optimized h-exchange being a more powerful neighborhood in terms of solutions quality.

3.4 Conclusion

We have presented the first systematic experimental analysis of heuristics for BAP along with some theoretical results on local search algorithms worst case performance.

Three classes of neighborhoods - h-exchange, $[h,p]$-exchange and shift based - are introduced. Some of the neighborhoods are of an exponential size but can be searched for an improving solution in polynomial time. Analysis of local optimums in terms of domination properties and relation to average value $\mathcal{A}(Q,C,D)$ are presented.

Several greedy, semi-greedy and rounding construction heuristics are proposed for generating reasonable quality solution quickly. Experimental results show that *RandomXYGreedy* is a good alternative among the approaches. The built-in randomized decision steps make this heuristic valuable for generating starting solutions for improvement algorithms within a multistart framework.

Extensive computational analysis has been carried out on the searches based on described neighborhoods. The experimental results suggest that the very large-scale neighborhood (VLSN) search algorithm - *Alternating Algorithm (AA)*, when used within multistart framework, yields a more balanced heuristic in terms of running time and solution quality. A variable neighborhood search (VNS) algorithm, that strategically uses optimized 2-exchange neighborhood and AA neighborhood, produced superior outcomes. However, this came with the downside of a significantly larger computational time.

We hope that this study inspires additional research work on the bilinear assignment model, particularly in the area of design and analysis of exact and heuristic algorithms.

Chapter 4

Partition Assignment Problem and Applications

In this chapter we present a previously unexplored model, named Partition Assignment Problem, and its connection to NSERC Engage project with the industrial partner, on the topic of ridesharing.

We first introduce this theoretical model that is based on combining assignments and set partitioning. We mention preliminary analysis of general Partition Assignment Problem (PAP) and variations, observed in practice. In later sections of the chapter we talk about the process of understanding the issues and challenges, presented to us by industrial partner. Next we describe the first approach we have tried to tackle the problem - matching. We discuss the shortcomings of the method and the reasoning behind shifting towards mathematical programming. In the later section we talk about how we extended the model to incorporate innate dynamism of the problem. We explain the origins of our data and some experimental outcomes, and conclude with potential directions to improve the performance.

4.1 Partition assignment problem

Let $N = \{1, 2, \ldots, n\}$ and $E = \{1, 2, \ldots, p\}$ be finite sets and $F = \{S_1, S_2, \ldots, S_m\}$ be a family of subsets of E. The index set of elements of F is denoted by $M = \{1, 2, \ldots, m\}$. For each $(i, j) \in N \times M$, a cost c_{ij} is prescribed. The value c_{ij} can be viewed as the cost of assigning the set S_j to i. Then the *partition assignment problem* (PAP) is to find a partition $\{S_{k_1}, S_{k_2}, \ldots, S_{k_r}\}$ of E such that $S_{k_i} \in F$ for $i = 1, 2, \ldots, r$ and an assignment of $S_{k_1}, S_{k_2}, \ldots, S_{k_r}$ to $\{1, 2, \ldots, n\}$ such that the total cost of assignment is as small as possible.

An integer programming formulation of this problem can be given as follows. For each $t \in E$ let $\Delta(t) = \{j : t \in S_j\}$. Consider the decision variables x_{ij} given by

$$x_{ij} = \begin{cases} 1 & \text{if } i \in N \text{ is assigned to the subset } S_j, \\ 0 & \text{otherwise.} \end{cases}$$

Then PAP can be formulated as a 0-1 integer programming problem:

$$\begin{aligned} \text{Minimize} \quad & \sum_{i \in N} \sum_{j \in M} c_{ij} x_{ij} & \\ \text{Subject to} \quad & \sum_{i \in N} x_{ij} \leq 1, \quad \forall j \in M & (4.1) \\ & \sum_{j \in M} x_{ij} \leq 1, \quad \forall i \in N & (4.2) \\ & \sum_{j \in \Delta(t)} \sum_{i \in N} x_{ij} = 1, \quad \forall t \in E & (4.3) \\ & x_{ij} \in \{0, 1\}, \quad \forall i \in N, \forall j \in M & (4.4) \end{aligned}$$

Note that in presence of equation (4.3), equation (4.1) is redundant and can be discarded. It may be advantageous to retain equation (4.1) in the model since it might provide a tighter continuous relaxation.

PAP can also be viewed as a bilevel program combining set partitioning and weighted bipartite matching as follows. Let $S^\alpha = \{S_{\alpha_1}, S_{\alpha_2}, \ldots, S_{\alpha_u}\}$ be a subfamily of F, such that S^α is a partition of E. Construct the complete bipartite graph $G^\alpha = (V^\alpha, H^\alpha)$ with the generic bipartition of $V^\alpha = N \cup U^\alpha$, where $U^\alpha = \{\alpha_1, \alpha_2, \ldots, \alpha_u\}$ and the cost of the edge $(i, \alpha_j) = c_{i\alpha_j}$. A matching in G^α is called *pseudo-perfect* if all nodes in U^α are matched. Let M^α be a minimum cost pseudo-perfect matching in G^α. Then the cost of the partition S^α is $C(S^\alpha) = \sum_{(i,j) \in M^\alpha} c_{ij}$. Note that $C(S^\alpha)$ is precisely the cost of the minimum cost pseudo-perfect matching in G^α. Thus PAP can be written as

$$\begin{aligned} & \text{Minimize} \quad C(S^\alpha) \\ & \text{Subject to } S^\alpha \in F' \end{aligned}$$

where F' is the collection of all partitions of E that are in F, that is, $F' = \{S^\alpha = \{S_{\alpha_1}, S_{\alpha_2}, \ldots, S_{\alpha_u}\} : S_{\alpha_i} \in F, 1 \leq i \leq u \leq n, \text{ and } S^\alpha \text{ is a partition of } E\}$.

4.1.1 Complexity

We now show that PAP is strongly NP-hard. The bottleneck version of PAP is obtained by replacing the objective function Minimize $\sum_{i \in N} \sum_{j \in M} c_{ij} x_{ij}$ in PAP with Minimize $\max_{i \in N, j \in M} \{c_{ij} x_{ij}\}$. The resulting problem is called ***bottleneck partition assignment problem*** (BPAP).

Theorem 30. *BPAP is strongly NP-hard.*

Proof. Follows from NP-hardness of the 3-partition problem. □

Corollary 31. *PAP is strongly NP-hard.*

Proof. It is well known that a bottleneck problem can be solved in polynomial time if the corresponding minsum problem can be solved in polynomial time [64]. Thus a polynomial time algorithm for PAP would yield a polynomial time algorithm for BPAP. The result now follows from Theorem 30. □

An alternative proof of Theorem 30 and Corollary 31 will be available from the NP-hardness of a different special case, presented in Section 4.2.

4.1.2 Average cost of solution

The number of different ways to partition a set of p elements is determined by the p-th *Bell number* B_p. The following summation formula represents B_p:

$$B_p = \sum_{k=0}^{p} \left\{ \begin{matrix} p \\ k \end{matrix} \right\} = \sum_{k=0}^{p} \frac{1}{k!} \sum_{g=0}^{k} (-1)^{k-g} \binom{k}{g} g^p \tag{4.5}$$

where $\left\{ \begin{matrix} p \\ k \end{matrix} \right\}$ - *Stirling number of the second kind*, is the number of different ways to partition a set of p elements into exactly k nonempty subsets.

Notice that given the partition $S^\alpha = \{S_{\alpha_1}, S_{\alpha_2}, \dots, S_{\alpha_u}\}$, the number of different ways to assign $S_{\alpha_1}, S_{\alpha_2}, \dots, S_{\alpha_u}$ to $\{1, 2, \dots, n\}$ is simply the number of u-permutations of n. Since we only consider partitions with $u \leq n$, we will need to substitute p in the summation by $\min\{n, p\}$. Hence, it is possible to count the number of different solutions to PAP by extending (4.5):

$$\begin{aligned} D_{n,p} = \sum_{k=0}^{\min\{n,p\}} \left\{ \begin{matrix} p \\ k \end{matrix} \right\} \frac{n!}{(n-k)!} = \sum_{k=0}^{\min\{n,p\}} \frac{1}{k!} \sum_{g=0}^{k} (-1)^{k-g} \binom{k}{g} g^p \frac{n!}{(n-k)!} = \\ = \sum_{k=0}^{\min\{n,p\}} \binom{n}{k} \sum_{g=0}^{k} (-1)^{k-g} \binom{k}{g} g^p \end{aligned} \tag{4.6}$$

Consider some particular assignment (i, j) of $i \in N$ to $S_j \in F$. Then, the number of different solutions to contain this assignment is exactly $D_{n-1,p-|S_j|}$. It can be derived as the number of different solutions to a smaller PAP with $N' = N \setminus \{i\}$ and $E' = E \setminus S_j$. Using this fact, the total cost of all different solutions to PAP is as follows:

$$T = \sum_{s_j \in F} \sum_{i=1}^{n} c_{ij} D_{n-1,p-|s_j|} = \sum_{s_j \in F} c_j D_{n-1,p-|s_j|} \tag{4.7}$$

where $c_j = \sum_{i=1}^{n} c_{ij}$ - sum of possible assignment costs for a particular subset $S_j \in F$.

Now, it is possible to find the average cost of a solution to PAP:

$$A = \frac{T}{D_{n,p}} = \frac{\sum_{s_j \in F} c_j D_{n-1,p-|s_j|}}{D_{n,p}} \tag{4.8}$$

4.2 Modelling ridesharing as partition assignment problem

Several combinatorial optimization problems of interest can be formulated under this general framework. This include allocation of customers to facilities, bus route allocation, less-than-truckload trailer truck assignment, multiprocessor scheduling problems, ride sharing problem, etc. Depending on the specific application scenario, minor changes from this general model may be required to represent the real application and to handle additional constraints. Our interest in PAP was primarily due to its application in ride sharing.

Let us first consider a simplified version of the problem called the *simple ridesharing problem* (SRP). We will see that SRP can be modeled as PAP. We have a set $N = \{1, 2, \ldots, n\}$ of drivers and a set $E = \{1, 2, \ldots, p\}$ of passengers. The drivers and passengers are traveling to the same destination. Each driver has a small capacity vehicle (eg. a car) and is willing to accommodate some passengers so that the capacity of the vehicle is not exceeded. We assume that the number of drivers are sufficiently large so that all passengers can be accommodated and the capacity of each vehicle is less than 5. The potential passenger groups are given by nonempty subsets of E with cardinality at most 3 (capacity excluding the driver) and we denote this family by $F = \{S_1, S_2, \ldots, S_m\}$, say. Thus, $|F| = \binom{p}{1} + \binom{p}{2} + \binom{p}{3} = O(p^3)$. Given a subset, say $S_j \in F$ and a driver $i \in N$, the minimal travel time a_{ij} required to pick up all the passengers by driver i and drive them to the destination can be calculated in constant time by simple enumeration. Additionally, let b_i be the minimal required travel time for driver i to get from origin to destination by himself. The corresponding cost of assigning driver i to subset of passengers S_j is then determined as $c_{ij} = a_{ij} - b_i$. Intuitively, not assigning any passengers to driver results in him going alone, and by defining c_{ij} in this way we incorporate the cost of this solo trip into objective function.

Theorem 32. *The SRP is NP-hard, even if maximal vehicle capacity is* 3 *and distances satisfy triangle inequality.*

It is shown by direct reduction from a special case of the two-to-one assignment problem. NP-hardness of this special two-to-one assignment problem is proven in Lemma 33 by reduction from the general two-to-one assignment problem, using a modification of the gadget found in several papers dealing with similar assignment problems, see [33, 56].

Before the proofs, definitions of three-dimensional and two-to-one assignment problems is presented.

Three dimensional assignment problem (3DA)
Input: Three disjoint sets X, Y, Z ($|X| = |Y| = |Z| = n$), and the costs c_{ijk} for all triples $(i, j, k) \in X \times Y \times Z$.
Goal: Find a set M of n mutually disjoint triples that minimize $\sum_{(i,j,k)\in M} c_{ijk}$.

The 3DA is NP-hard, even if the costs c_{ijk} can take only two distinct values. See [50] for a proof.

Two-to-one assignment problem (2-1AP)
Input: Two disjoint sets X, Y ($2|X| = |Y| = 2n$), and the costs c_{ijk} for all triples $(i, j, k) \in X \times Y \times Y$.
Goal: Find a set M of n mutually disjoint triples that minimize $\sum_{(i,j,k)\in M} c_{ijk}$.

The 2-1AP is also NP-hard, even if the costs c_{ijk} can take only two distinct values. Namely, it contains the 3DA as a special case: Starting from an 3DA instance $I' = (X', Y', Z', \{c'_{ijk}\})$ where $c'_{ijk} \in \{a, b\}$, $a < b$, one can obtain an equivalent instance of the 2-1AP $I = (X, Y, \{c_{ijk}\})$ by setting $X := X'$, $Y := Y' \cup Z'$ and defining costs c_{ijk} to be equal to corresponding c'_{ijk}, extended by setting the costs of new feasible triples to be equal to b. Approximability of the 2-1AP is investigated in [56].

Next we proof NP-hardness of a special case of the 2-1AP. Let there be a (symmetric) distance d_{ij} between any two points from $X \cup Y$, and let the cost of a triple be defined by

$$c_{ijk} := \min\{d_{ij} + d_{jk}, d_{ik} + d_{jk}\} \qquad i \in X,\ j, k \in Y.$$

We denote this problem by **S2-1AP**.

Lemma 33. *The S2-1AP is NP-hard, even if the distances d_{ij} can take only values* 1 *and* 2.

Proof. We prove the lemma by reducing 2-1AP to S2-1AP. Let $I' = (X', Y', \{c'_{ijk}\})$ with $c'_{ijk} \in \{1, 2\}$, be an instance of 2-1AP. Let T be a set of triples (x, y', y'') for which $c'_{xy'y''} = 1$, and let $m := |T|$. For every $t = (x, y', y'') \in T$ we define a gadget $G(t)$ to be a set of six auxiliary points $\{x_1(t), x_2(t), y_1(t), y_2(t), y_3(t), y_4(t)\}$ and corresponding eight edges depicted by Figure 4.1.

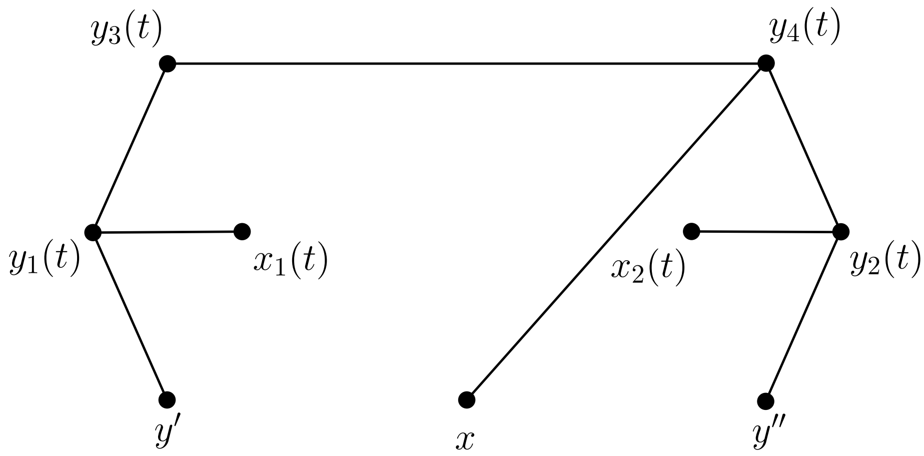

Figure 4.1: Gadget $G(t)$ for a triple $t = (x, y', y'')$

Now we build an instance $I = (X, Y, \{d_{ij}\})$ of the S2-1AP as follows: $X = X' \cup \{x_i(t) \colon t \in T,\ i = 1, 2\}$, $Y = Y' \cup \{y_i(t) \colon t \in T,\ i = 1, 2, 3, 4\}$, and d_{ij} is equal to 1 if there is an edge between i and j in some $G(t)$, $t \in T$, and 2 otherwise. Note that $|X \cup Y| = 3n + 6m$.

It is not hard to see that the optimal solution value for I is $2n + 4m$ if and only if the optimal solution value for I' is n. Namely, note that points in a gadget $G(t = (x, y', y''))$ can be partitioned into triples of cost 2 only in two ways: by triples $(x_1(t), y', y_1(t))$, $(x_2(t), y'', y_2(t))$, $(x, y_3(t), y_4(t))$ or by triples $(x_1(t), y_1(t), y_3(t))$, $(x_2(t), y_2(t), y_4(t))$. The first case corresponds to taking triple (x, y', y'') into the solution of I', and the second case in not taking it into the solution. $\square$

$G(t)$ is a modification of the gadget (first) used in [50] to establish NP-hardness of the PARTITION INTO TRIANGLES.

A SRP instance is given by set of drivers R, set of passengers P, and the destination t. Every driver $r \in R$ has its capacity $q(r)$, which is the number of passengers that he/she can take. We assume that $\sum_{r \in R} q(r) = |P|$. Furthermore, there is a distance ℓ_{ij} of the direct path between $i, j \in R \cup P \cup \{t\}$. The goal is to assign all passengers to drivers with capacities being satisfied, so that sum of shortest paths from drivers through its assigned passengers ending in destination t, is minimized.

Proof of Theorem 32. We do the reduction from the S2-1AP with $d_{ij} \in \{1, 2\}$. Let $I = (X, Y, \{d_{ij}\})$ be such instance of the S2-1AP. We build an instance of the SRP by setting $N := X$ to be the set of drivers, $E := Y$ to be the set of passengers. All vehicle capacities are set to be 3. All distances between drivers and passengers, and two passengers are set to be d_{ij}. All other distances (distances to the destination t) are set to be 1.

Observe the value of the shortest path from driver i, through passengers $S_j = \{p_1, p_2\}$, ending in destination t. It is equal to

$$a_{ij} = \min\{d_{ip_1} + d_{p_1p_2} + 1, \ell_{ip_2} + \ell_{p_2p_1} + 1\}.$$

Therefore, the cost c_{ij} of assigning $S_j = \{p_1, p_2\}$ to i is equal to

$$c_{ij} = \min\{d_{ip_1} + d_{p_1p_2}, d_{ip_2} + d_{p_1p_2}\}.$$

Hence, covering our passengers with shortest paths from drivers to the destination, is equivalent to solving the starting instance I of the S2-1AP.

Lastly, note that all distances between drivers, passengers and the destination in our SRP instance have values 1 or 2, hence the triangle inequality is satisfied. □

Thus SRP is precisely a PAP. We can associate a bipartite $G = (V, L)$ where the generic bipartition of V is $N \cup M$ where $M = \{1, 2, \ldots, m\}$.

Depending on some constraints the size of G may be reduced. For example, if a set S_j contains two passengers who may not want to travel together. Such sets are dropped from F and hence the corresponding node $j \in M$ from G is removed. Let d_i be the time required to reach destination by driver i without any additional passengers. Then some drivers may have constraints that they do not want to consider passenger group, say S_j with corresponding driving time c_{ij} exceeds a prescribed threshold. In such cases, the arc (i, j) is removed from G. Thus G need not be a complete bipartite graph. Most of the time, the driver time constraints make it really sparse. The case, when some of the passengers are allowed not to be assigned at all, is trivially modeled by adding dummy personal drivers for these passengers. Let us disregard this option for now and continue with the general setting. We now discuss the integer programming formulation given for PAP where G is not necessarily complete bipartite.

For each node i, we denote $A(i)$ the set of nodes that are adjacent to i. Then the integer programming formulation for SRP can be given as follows:

$$
\begin{aligned}
\text{Minimize} \quad & \sum_{i \in N} \sum_{j \in A(i)} c_{ij} x_{ij} \\
\text{Subject to} \quad & \sum_{i \in A(j)} x_{ij} \leq 1, \quad \forall j \in M && (4.9) \\
& \sum_{j \in A(i)} x_{ij} \leq 1, \quad \forall i \in N && (4.10) \\
& \sum_{j \in \Delta(t)} \sum_{i \in A(j)} x_{ij} = 1, \quad \forall t \in E && (4.11) \\
& x_{ij} \in \{0, 1\}, \quad \forall (i, j) \in L && (4.12)
\end{aligned}
$$

As discussed earlier, constraints (4.9) are redundant and may be removed from the formulation.

In SRP we assumed that all drivers must be using their cars to get to the destination. However, in practice some of these drivers may like the option to be a passenger in another vehicle. The ride-sharing problem with this flexibility is called the *flexible ridesharing problem* (FRP). We now observe that FRP can be formulated as SRP with slightly increased problem size. Let N_1 be the set of drivers who are willing to drive but also willing to go as a passenger in another vehicle and N_2 be the set of drivers who definitely want to drive. Note that $N = N_1 \cup N_2$. Without loss of generality assume $N_1 = \{1, 2, \ldots, n_1\}$ and $N_2 = \{n_1 + 1, n_1 + 2, \ldots, n\}$. Let $N_1' = \{p + 1, p + 2, \ldots, p + n_1\}$ and $E' = E \cup N_1'$. Note that the element $p + i$ in N_1' corresponds to the driver $i \in N_1$ viewed as a passenger. Let $F' = \{S_1', S_2', \ldots, S_{m_1}'\}$ be a family of feasible subsets of E' and $M' = \{1, 2, \ldots, m_1\}$. We restrict $|S_j'| \leq 3$ if it does not contain any elements of N_1' and $|S_j'| \leq 4$ if it contains at least one element of N_1'. Construct the bipartite graph $G' = (V', L')$ where the generic bipartition of $V' = N \cup M'$. Each node $i \in N_2$ is connected to a node j in M' where the set S_j is acceptable for the driver i. Respectively, a node $i \in N_1$ is connected to a node j in M' if S_j is acceptable for i, and $p + i \in S_j$. Similarly to SRP, given a driver $i \in N$ and a passenger subset $S_j \in F'$, let a_{ij} be the minimal travel time required to pick up all the passengers by the driver and ride them to the destination, and let b_i be the minimal required travel time for i to get from origin to destination alone. We derive $c_{ij} = a_{ij} - b_i - \sum_{k \in S_j \cap N_1' \setminus \{p+i\}} b_{k-p}$ as the assignment cost for our formulation. For each node i, let $A(i)$ be the collection of nodes in G' adjacent to i. Now, on G' we can write an integer program similar to that of SRSP, which will solve the FRSP.

4.2.1 Bipartite and weighted matching models

As the initial approach we have decided to use bipartite matching. Here we model our problem as maximum cardinality bipartite matching and solve it as maximum flow problem.

Arbitrarily select set of drivers D, $D \subseteq \{i : i \in R, m_i > 1\}$. P - set of passengers, $P = R \setminus D$.

For any driver $j \in D$ and passenger $k \in P$ define $s_{j,k}$ as travel cost savings of matching j and k together.

$$\begin{aligned} s_{j,k} = (c(l_j^+, l_k^+) + c(l_k^+, l_k^-) + c(l_k^-, l_j^-)) - (c(l_j^+, l_j^-) + c(l_k^+, l_k^-)) = \\ = c(l_j^+, l_k^+) + c(l_k^-, l_j^-) - c(l_j^+, l_j^-) \end{aligned} \tag{4.13}$$

Define capacity function C in the following way:

- $C(source, j) = m_j - 1, \forall j \in D$
- $C(j, k) = 1$ if j and k can travel together (satisfying time and maximum detour constraints), $\forall j \in D \, \forall k \in P$
- $C(k, sink) = 1, \forall k \in P$
- $C(\cdot, \cdot) = 0$ otherwise

Find maximum flow f through this network, and assign passenger k to driver j if $f(j, k) = 1$.

The issues of modelling our problem as matching were the following: Choice of drivers is arbitrary but influences heavily the resulting performance. We are not guaranteed to find a matching that yields minimum total travel cost. Pairwise compatibility checks approach does not guarantee that the final assignment of passengers to driver will satisfy time constraints or even reduce the total travel cost. To deal with some of the issues of this model we modified the edges with weights to make it a weighted matching and solve it as an assignment problem.

Arbitrarily select $D \subseteq \{i : i \in R, m_i > 1\}$. $P = R \setminus D$. The cost of matching driver j and passenger k is $s_{j,k}$ defined in (4.13).

We construct bipartite graph in the following way:

- left side of the graph consists of vertices corresponding to j, each copied $m_j - 1$ times, and $|P|$ "zero" vertices (for empty match possibility)
- on the right side of the graph there are vertices that correspond to k
- the edge weight between vertices u and v is $\begin{cases} s_{j,k} & \text{for } u \text{ corresponding to } j \\ 0 & \text{if } u \text{ is "zero" vertex} \end{cases}$

Solve assignment problem on the described graph to get assignment of passengers to drivers. "zero" vertex assignment means that passenger is not assigned to any driver.

The heuristic nature of the solution and the restriction to pairwise matching suggested to look for a more flexible model. We decided to try mathematical programming.

4.2.2 Integer linear programming formulation

Now we model our problem as integer linear program and use general purpose ILP solver.

Define carpool λ as a tuple $(j, \{k_1, k_2, \dots, k_q\})$, where $j \in R$ represents driver, and $\{k_1, k_2, \dots, k_q\} \subset R$ is a set of passengers that are going with him. We call λ feasible carpool if:

- $j \notin \{k_1, k_2, \dots, k_q\}$
- $0 \leq |\{k_1, k_2, \dots, k_q\}| \leq m_j - 1$
- there exists a path that starts at l_j^+, finishes at l_j^-, visits all the passenger origins and destinations in the proper order, and respects time constraints of everyone
- define $w(\lambda)$ to be a cost of the shortest path (as described above). $w(\lambda)$ must satisfy $w(\lambda) < c(l_j^+, l_j^-) + \sum_{p=1}^{q} c(l_p^+, l_p^-)$

Define Λ to be a set of all feasible carpools.

Variables: x_z - binary: 1 if carpool $\lambda_z \in \Lambda$ is used, 0 otherwise.

$$\min \sum_{\lambda_z \in \Lambda} w(\lambda_z) x_z \tag{4.14}$$

subject to

$$\sum_{\lambda_z \in \Lambda : i \in \{j\} \cup \{k_1, k_2, \dots, k_q\}} x_z = 1, \qquad \forall i \in R \tag{4.15}$$

4.2.3 Experimental results

The approach discussed above allowed us to get the exact solution to our problem. However we had to consider scalability issue for larger instances of the problem (in terms of the number of requests). We were able to solve instances up to 1000 requests within the reasonable time, which led us to think that integer programming is promising to try for the dynamic version of the problem.

Dynamism of the problem. In dynamic version we assume that requests come to us throughout the day. Using results described in previous section we decided to extend our integer programming model to incorporate this change. We have used rolling horizon approach thought the day to determine the set of requests to perform scheduling. Every fixed number of minutes the system analyzes the requests received since the last run, and tries to schedule them into carpools using integer programming.

Data generation and results. We generate our data for the full 24 hour day, based on statistics that were provided. Destination of each request is SFU Burnaby campus, and origins are spread around Vancouver metro area and Lower Mainland.

The input of the data generator is the following: number of requests; ratio of requests with a car (set to 30% based on statistics provided by company); hourly SFU Burnaby traffic distribution for average day (Fig. 4.2); distribution of requests origins among postal code areas for average day (Fig. 4.3).

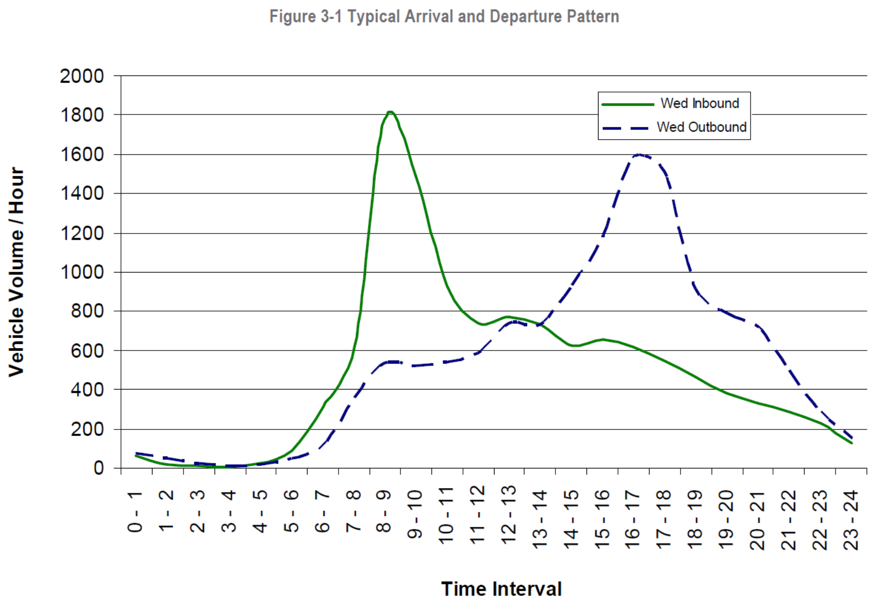

Figure 4.2: Hourly distribution of incoming and outgoing personal vehicle traffic at SFU Burnaby

Car capacity for requests with available cars is set to 4 seats. Time windows are generated such that the latest arrival time t_r^{end} of request r follows the distribution from Fig. 4.2, and earliest departure time t_r^{begin} is randomly selected in the following way:

$$t_r^{\text{begin}} \in [t_r^{\text{end}} - max\{2b_r, 3600\}, t_r^{\text{end}} - b_r] \tag{4.16}$$

Here b_r is the cost (in seconds) of the shortest path from origin to destination. In this way the feasibility of all single trips is maintained.

We solve the following instance with our dynamic scheduler: 1000 total requests for 24 hour day period, 300 are from users with cars (4 seats total in each car), 700 are from users without cars (transit users). Out of those 1000, we assume that we have a knowledge of 500 requests in the evening of the previous day. Remaining 500 requests show up dynamically throughout the day (we assume that these requests are submitted by user exactly 1 hour before the user can leave his destination, i.e. earliest possible departure time). Latest possible arrival time of each request is chosen according to the time distribution that was

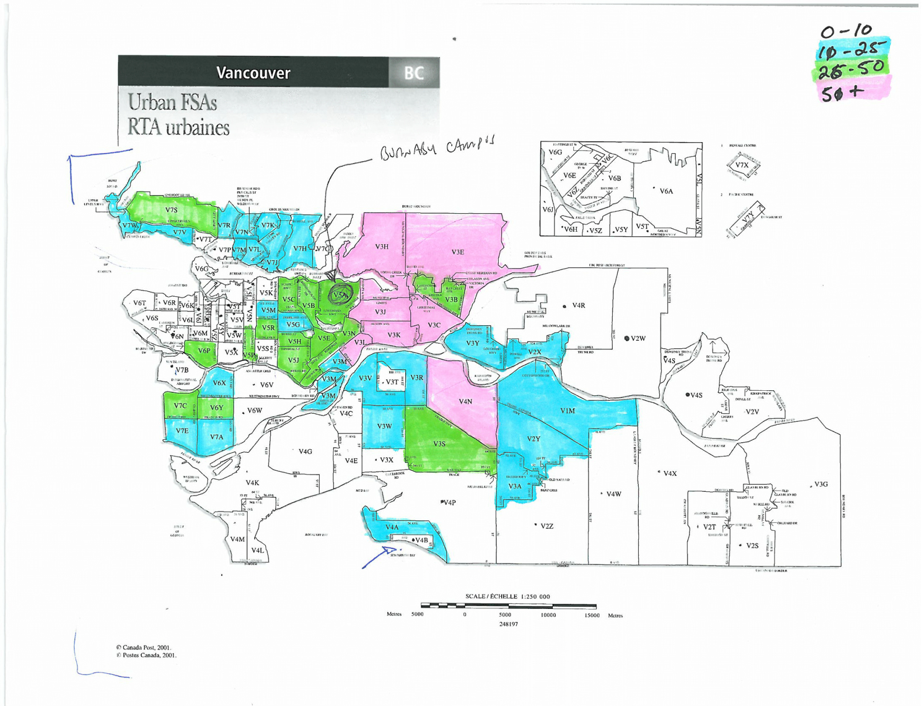

Figure 4.3: Provided distribution of carpooling requests by postal zones (FSA districts) for a typical day

provided. Earliest possible departure time is then generated to be such, that the user have enough time to get to SFU by his deadline, plus some random ahead time of at most the time it takes for him to do the trip alone (so if I need to be at SFU by 9:00, and it takes me 1 hour to go directly, then my earliest possible departure time will be between 7:00 and 8:00).

Maximum detour time (carpool trip overhead compared to direct solo trip) is set to be 10 minutes. The cost of transit user going alone (not being carpooled with anyone) is 10% of the corresponding car route that would have taken him from his home to the destination. Destination is SFU Burnaby for everyone. Origins are spread out Vancouver metro area. We run our scheduler every 30 minutes. The system takes gathered over this period of time new requests together with requests that we have already been aware of, and produces the optimal schedule. The time after which we are not allowed to change the scheduled carpool is set to be 30 minutes before the time the driver is supposed to leave his origin. At that point these driver and passengers are out of the scheduler, as we have committed to using this carpool.

We have obtained the following results for these instances. 600 out of 700 transit users and 39 out of 300 car users are not matched by the system. 100 remaining transit users and 261 car owners were matched together into 48 carpools with 2 people, 44 carpools with 3 people, and 31 carpools with fully packed cars (4 people). Computational time required for each run of the scheduler varies from 2 minutes in the beginning of the day (when more requests are known ahead and not many have been committed) to few seconds in the end of the day (when most requests are dealt with already).

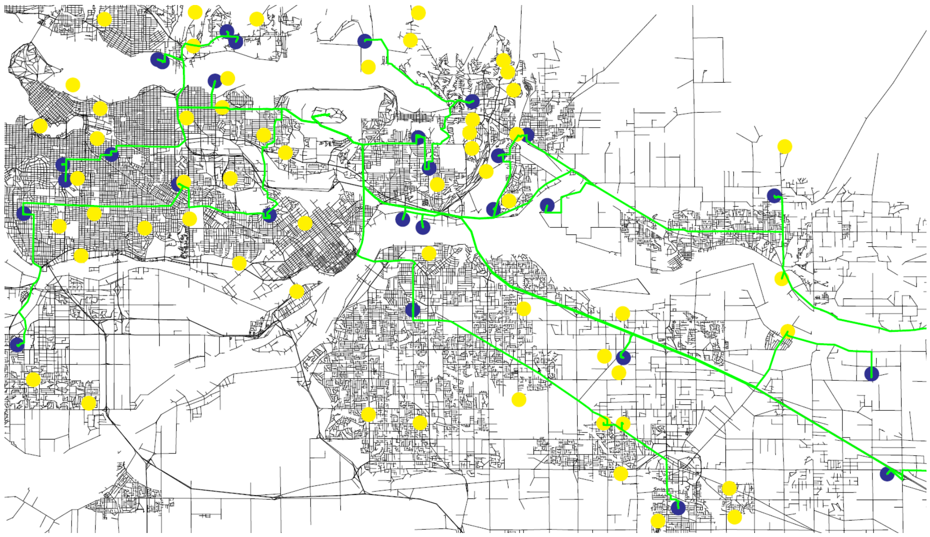

Figure 4.4: An example of the static ridesharing problem solution for an instance of size 100

4.3 Conclusion

In this chapter we have explored the problem that arises in ridesharing systems, and can formally be stated as an assignment of vehicles to groups of passengers that should carpool together. Among others we have explored a formulation that relies on assignment combinatorial structure, and is described as the Partition Assignment Problem. We presented some complexity results on the problem, including reductions to and from other famous assignment problems. An integer programming formulation is used to obtain solutions to generated instances of various sizes.

This versatile model has not been studied before and therefore presents many areas of open problems such as: analysis of special cases, developing constant factor approximations, further research of domination properties of local search methods, developing heuristics and exact algorithms for the problem.

Chapter 5

Laboratory Samples Delivery

The results of work discussed in this chapter were presented at International Conference on Operations Research and Enterprise Systems (ICORES) 2015 in Lisbon, Portugal [111].

Every metropolitan city has hospitals of varying sizes, each cost-effective in serving the healthcare needs of its surrounding population. Most hospitals include a laboratory that can perform a variety of tests on samples collected from its patients. Since laboratories in smaller hospitals are often not equipped to perform all tests on its samples, these samples have to be sent to laboratories in larger hospitals for testing. This paper addresses the problem of routing samples to hospitals, called the Hospital Laboratory Courier Routing Problem (HLCRP). This is the pickup and delivery problem with time windows without capacity constraints and with transshipments allowed. Even though we address the specific problem of routing test samples between hospitals, our model and solution procedure can be applied to other problems.

The test samples are collected from the patients in the hospitals during a day. The samples include blood, urine, sputum, or tissue, and each sample has a deadline before which the test should be conducted. The hospitals are located in a given geographical area, such as the metropolitan area of a city. Each hospital is equipped with a laboratory of a given capability. Some samples can be tested at the hospital where it was collected, while others have to be transported to another hospital with better equipped laboratories.

We consider three levels of capability for the laboratories: Level 1, Level 2, and Level 3. The Level 3 laboratories are the most well-equipped of the laboratories, and can perform all the tests on a sample. The Level 1 laboratories are the least equipped, with the Level 2 laboratories capable of performing all the tests that Level 1 laboratories can perform, as well as additional tests. In the geographical area we consider in this paper, there is only one Level 3 laboratory, two Level 2 laboratories, eleven Level 1 laboratories, and three locations without any laboratory.

Depending on the tests required to be performed on a sample, the level of the laboratory is specified together with the deadline. Currently, in practice the destination hospital is

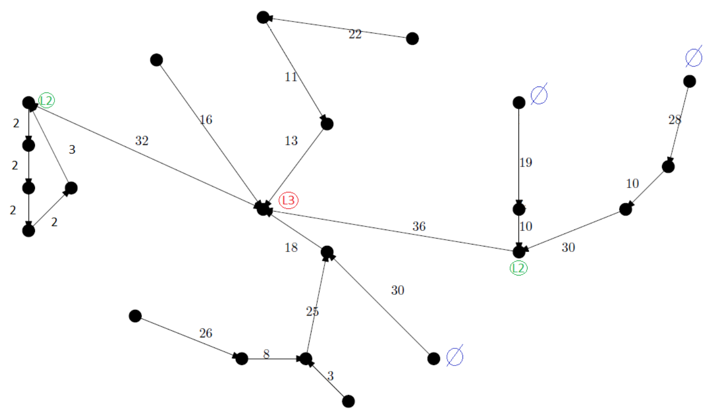

Figure 5.1: Typical map layout for HLCRP

explicitly specified by the hospital staff. Thus, each sample has its origin, its destination, and the deadline before which the test should be performed.

The transportation of samples is done by a fleet of vehicles of a fixed size. In addition, the use of taxis is also allowed to transport samples with the impending deadlines. For the fleet of vehicles, there is no depot, and each vehicle can start its route at any hospital and finish it at any other hospital. (This characteristic of the problem also comes from the situation observed in practice where the cost of the route does not include the cost of getting the vehicle to the first hospital in the route.)

In practice, the deadline is a hard constraint and is the driving force in determining the routes and schedules. The samples are small enough that all of them can fit easily into a small car. Thus, in our model we do not consider capacity constraints on the vehicles.

The Hospital Laboratory Courier Routing Problem (HLCRP) deals with finding the routes and schedules for the vehicles such that all the samples are transported within their deadlines, and a linear function of the total distance travelled by the vehicles and taxis is minimized. Since the cost of using a taxi is several times higher than the standard vehicle, the optimizer should also reduce the number of taxi calls. In addition to this objective, we also address the objective of minimizing the number of taxis used to transport the samples.

Thus, the HLCRP is a variation of the pickup and delivery problem with time windows, where capacity constraints are not considered and transshipment is allowed ([120], [97]). Problems that address pickup and delivery of people using transshipment (called transfers) have also been addressed [32].

The start of the time window for a sample is the time when the sample is collected from the patient. The deadline is the time by which the sample has to reach the designated laboratory in order to be processed on time. The time windows are hard constraints (each sample has to be processed within its deadline). Transshipment is allowed because there is no requirement that a sample has to be delivered by the same vehicle that picked up the sample. Thus, a sample can be moved from one vehicle to another at an intermediate hospital.

Finally, if a sample cannot be transported by a vehicle in the fleet within its deadline, a taxi can be used to transport the sample. The HLCRP can be used to model many problems that may arise in other domains that involve routing and scheduling.

Our case study considered the case of a region in Canada that sub-contracts lab sample transportation to a courier company. The planning of the routes are done two times a year and the routes are fixed for that period. During that time period, for day-to-day operations when urgent sample transporation requests has to be served, or when the regular routes would not be enough to serve lab transportation requests on time, a taxi cab is used.

Each sample that requires transportation defines one transportation request.

There is no restriction that the sample has to be picked up and delivered by the same vehicle. The samples can be moved from one vehicle to another at any intermediate hospital site. Also, the samples can be removed from one vehicle at an intermediate hospital location and left there to wait for another vehicle that will take the sample to another intermediate hospital or to its destination hospital.

The problem is the pickup and delivery problem with time windows and with transshipment. The start of the time window of a sample is the time sample is collected from the patient. The deadline is the time the sample has to reach the designated lab in order to be processed on time. The time windows are the hard constraints. All samples has to be processed on time, and the re-taking of the sample from the patient has to be avoided at any cost.

Although the problem is dynamic, we consider the static version of the problem where the routes are generated in advance based on expected demand. Thus, in this paper, we are considering the static version of the problem. The sample transporation requests are satisfied by the courier fleet of vehicles that is dedicated to this transportation service. There is no depot. The vehicle can start and finish its route at any hospital. The start and the end location do not have to be the same. In addition, if the vehicles cannot serve a sample transportation request on time, a taxi will be called in to perform the trasnporation task.

In our problem, the objective is to minimize the total transportation costs that consists of the courier vehicle costs and the taxi costs. The transportation costs are proportional to the distance travelled.

The HLCRP may either be modeled as a vehicle routing problem (VRP), or as a multi-commodity network flow problem (MCNFP). Typically, the number of locations in the VRP is large, and each location has to be visited once. In contrast, in the MCNFP, the number of locations is smaller, though the number of requests originating at each location is large, with multiple visits to the same location during the day.

There is a large body of research on the VRP, with many surveys, including [31, 55, 92, 93, 127]. There are also many surveys on the time-constrained version of the VRP - the Vehicle Routing Problem with Time Windows (VRPTW) - including [16] and [75]. Examples of the methods for optimally solving the VRPTW include Desrochers et. al [40], who pioneered the column-generation approach for the vehicle routing problem. They decomposed the problem into a master problem and a subproblem, and solved the master problem using column generation. Kohl et. al [82] introduced cuts to the decomposition-based approach, and Kohl and Madsen [83] develop a Lagrangean relaxation approach to solve the VRPTW exactly. For a comprehensive review of using the column generation method to solve the VRPTW, see [75].

We model the HLCRP as an MCNFP. Each node in the network is a hospital at a particular time instant, and each arc between two nodes is the route between the corresponding hospitals. Each set of boxes that are carried together by a vehicle is a commodity that flows through the network. Such models have been used to design networks and routes for public transportation [24], to solve ship routing and scheduling problems [29], in maritime transportation [17], airline schedule planning [57], and ferry scheduling [77].

In related work, heuristics using genetic algorithms have been used to solve the problem of routing blood samples collected from hospitals and health care centres to two central laboratories in Spain [58]. In this problem, in addition to imposing time windows on samples, vehicles also have capacity restrictions. Finally, [112] provides a comprehensive survey of operations research methods used in the healthcare industry. The applications listed in the survey are far too many to list here.

5.1 Modelling the problem as multi-commodity network flow

We modeled the HLCRP as a multi-commodity network flow problem. Each hospital is represented by a set of nodes at regularly spaced time instants.

In the model we use for HLCRP, the time horizon is divided into intervals of size δ (where δ is a suitably chosen constant), and each hospital is represented by multiple nodes, one for each time instant (the multiple of δ). Representing each node by multiple nodes, one for each time instant, is a standard modelling approach, usually called time expanded network. This approach has been successfully used to solve many practical instances of similar routing and scheduling problems.

A directed arc exists between two nodes a and b, if it is feasible to travel from node a to node b within the corresponding time. The movement of the samples and the vehicles represent the flow through the network.

Since in practice, the hospital staff packs samples with the same destination hospital and close deadlines into one package, we consider the problem of routing the packages. (The difference between the deadlines can be specified by providing a threshold.) Each package has a destination and a deadline.

The fleet of vehicles is of a fixed size. Since we observed in practice that the vehicle capacities are not a restriction, we can assume that the problem does not have capacity constraints. There is no depot, i.e., a vehicle can start and finish its route at any site node at any time, with no additional cost. In other words, the vehicles do not need to start from and return to a depot.

The solution to our problem consists of a set of routes and schedules. Each route is a sequence of hospital locations, each of which has the arrival time and the departure time. We assume that each vehicle starts immediately from the first pick-up point, thus there is no travel cost from and to the depot.

We assume that samples that have the same destination and deadline are transported in a package (each package now has a destination, as well as a deadline).

We also consider a second set of vehicles - the taxis. There are no limits on the number of taxi trips. However, using a taxi to travel between two nodes costs ρ times more than using a vehicle.

We provide details of the network construction for the model in Section 5.1.1, and the mathematical programming formulation for the model in Section 5.1.2.

5.1.1 Network Construction

Time Discretization:

The size of the network is a function of the discretization time, denoted Δt. Δt is specified in minutes. T is the set of all discrete time instants/stamps, M is the set of all packages, and l_j^p, $l_j^d \in N$ denote the pickup, delivery locations of package j, $\forall j \in M$. Furthermore, t_j^p, t_j^d denote the earliest pickup time, latest delivery time of package j, $\forall j \in M$, and $h^b = \min_{j \in M} t_j^p$, $h^e = \max_{j \in M} t_j^d$ denote the beginning, end of the horizon for our problem.

$|T|$ is given by $|T| = \lfloor \frac{h^e - h^b}{\Delta t} \rfloor + 1$. The earliest pickup time stamp of package $j \in M$ is given by $\tau_j^p = \lceil \frac{t_j^p - h^b}{\Delta t} \rceil$. The latest delivery time stamp of package $j \in M$ is given by $\tau_j^d = \lfloor \frac{t_j^d - h^b}{\Delta t} \rfloor$. The discretized cost (distance) from $u \in N$ to $v \in N$ (measured in time stamps), denoted $\delta_{u,v}$, is given by $\delta_{u,v} = \lceil \frac{d_{u,v}}{\Delta t} \rceil$. Thus $\delta_{u,u} = 1$ denotes waiting for one time stamp at node $u \in N$. We assume the graph G is not complete, so the distance $d_{u,v}$ (as well as the discretized distance $\delta_{u,v}$) is ∞ if there is no direct route from node u to node v. We let $\sigma_{u,v}$ denote the shortest path distance from u to v in the network (computed in units of δ).

Nodes and Arcs in the Network:

We are given a set N of site nodes (each site node denotes a hospital or test site). Corresponding to each site node $u \in N$, we construct q copy nodes $(u, 1)$,$(u, 2)$,..., (u, q), where (u, l) represents the copy of site node u at time stamp l. We also add a start node s and a destination node f to the set of copy nodes. We now describe the set of arcs comprising the network.

We have three types of arcs in the network, the set of package arcs A_p, the set of vehicle arcs A_c, and the set of taxi arcs A_t. We provide the ability to use additional problem-specific information to reduce the number of arcs (and therefore the size) of our model. Thus, if no package travels from node (u, q) to node (v, r) in any optimal route, then there is no arc between nodes (u, q) and node (v, r). Arcs may also not be present if routing a package through the arc violates feasibility. To simplify exposition, we refer to a copy node as a node in rest of the paper.

A package arc between two copy nodes indicates that a package can travel between the corresponding site nodes feasibly in time. Thus, the package arc $e^j_{(u,q),(v,r)}$ is in set A_p if package j can arrive at site node u before time q, can leave site node v at or after time r, and be feasibly delivered at its destination node before its deadline. Moreover, r is the earliest possible time during which the package may arrive at site node v after departing from site node u at time p. Similarly, a vehicle arc $e^c_{(u,q),(v,r)}$ (taxi arc $e^t_{(u,q),(v,r)}$) exists if a vehicle (taxi) can feasibly travel from copy node (u, q) to copy node (v, r).

We describe below the conditions that have to be fulfilled for the existence of package arcs, vehicle arcs, and taxi arcs. We define a boolean variable $b_{u,v}$ which is set to 1 if a package originating at copy node (u, q) is allowed to travel through copy node (v, r) (it is set to 0 otherwise). This boolean variable is used to specify the conditions for the existence of package arcs.

Conditions for the existence of package arcs:
$\forall j \in M, \forall u, v \in N \quad u \neq l^d_j \quad v \neq l^p_j, \forall q \in T$, package arc $e^j_{(u,q),(v,r)} \in A_p$ if each of the conditions below hold:

$$b_{l^p_j,u} = 1 \wedge b_{l^p_j,v} = 1 \tag{5.1}$$

$$q \geq \tau^p_j + \sigma_{l^p_j,u} \tag{5.2}$$

$$r = q + \delta_{u,v} \leq \tau^d_j - \sigma_{v,l^d_j} \tag{5.3}$$

Here, Equation (5.2) (respectively (5.3)) determine the earliest possible departure time of the package from u (respectively the latest possible arrival time at v). Note that there can be more than 1 package arc (for different packages) between copy nodes (u, q) and (v, r).

Conditions for the existence of vehicle and taxi arcs:

$\forall u, v \in N \quad \forall q \in T \quad e^c_{(u,q),(v,r)} \in A_c$ $(e^t_{(u,q),(v,r)} \in A_c)$ if $r = q + \delta_{u,v} \leq |T| - 1$
$\forall v \in N \quad \forall r \in T \quad e^c_{s,(v,r)} \in A_c$
$\forall u \in N \quad \forall q \in T \quad e^c_{(u,q),f} \in A_c$

These constraint ensure that there is a feasible arc between two locations with respect to travel time. In addition, we add a vehicle arc from the start node s to every copy node (u, q) and from every copy node (v, r) to the end node f.
We are now ready to provide the formulation.

5.1.2 Mathematical Programming Formulation

The decision variables are the boolean variables $x_{j,u,q,v,r}$, $y_{u,q,v,r}$, and $z_{u,q,v,r}$, that indicate whether a package, vehicle, or taxi travels along arc $e^j_{(u,q),(v,r)}$, $e^c_{(u,q),(v,r)}$, or $e^t_{(u,q),(v,r)}$, respectively. The number of vehicles used is modeled using integer variables $s_{v,r}$ and $f_{u,q}$.

$$\min \sum_{u,q,v,r:\, e^t_{(u,q),(v,r)} \in A_t} d_{u,v}(y_{u,q,v,r} + \rho z_{u,q,v,r}) \tag{5.4}$$

Subject to

Package pickup constraints (ensures each package is picked up):

$$\sum_{q,v,r:\, e^j_{(l^p_j,q),(v,r)} \in A_p} x_{j,l^p_j,q,v,r} = 1 \qquad \forall j \in M \tag{5.5}$$

Package delivery constraints (ensures each package is delivered):

$$\sum_{u,q,r:\, e^j_{(u,q),(l^d_j,r)} \in A_p} x_{j,u,q,l^d_j,r} = 1 \qquad \forall j \in M \tag{5.6}$$

Package transit constraints (ensures if a package enters a copy node, it also exits the copy node):

$$\sum_{u,q:\, e^j_{(u,q),(v,r)} \in A_p} x_{j,u,q,v,r} = \sum_{w,s:\, e^j_{(v,r),(w,s)} \in A_p} x_{j,v,r,w,s} \qquad \forall j \in M, \forall v \in N \setminus \{l^p_j, l^d_j\}, \forall r \in T \tag{5.7}$$

Package carry constraints (ensures packages must be carried by vehicles or taxis):

$$y_{u,q,v,r} + z_{u,q,v,r} \geq x_{j,u,q,v,r} \qquad \forall j, u, q, v, r :\ u \neq v \wedge e^j_{(u,q),(v,r)} \in A_p \tag{5.8}$$

Vehicles start constraint (ensures k vehicles start their routes):

$$\sum_{v,r\colon e_{s,(v,r)} \in A_c} s_{v,r} = k \tag{5.9}$$

Vehicles finish constraint (ensures k vehicles finish their routes):

$$\sum_{u,q\colon e_{(u,q),f} \in A_c} f_{u,q} = k \tag{5.10}$$

Vehicles transit constraints (ensures that the drivers that enter a copy node exit the copy node):

$$s_{v,r} + \sum_{u,q\colon e_{(u,q),(v,r)} \in A_c} y_{u,q,v,r} = f_{v,r} + \sum_{w,s\colon e_{(v,r),(w,s)} \in A_c} y_{v,r,w,s} \qquad \forall v \in N, \forall r \in T \tag{5.11}$$

Variables:

$$x, z \in \{0, 1\} \quad y, s, f \in \{0, 1, \dots, k\} \tag{5.12}$$

5.2 Implementation and Experimental Results

We solve the mathematical programming model above using the general-purpose mixed-integer program solver CPLEX. Since the time-space network can be too large, we apply arc reduction procedures prior to the integer programming formulation construction. Moreover, we remove some of the arcs in the network based on the available heuristic information about the general structure of possible routes. For example, in the graph networks corresponding to city maps, there are tendencies for routes to extensively use arterial roads. Another type of arc reduction can come from the fact that there is rarely a need for a package, that has its pickup and delivery in the same local area, to be travelling through another distant area of the network. All this extra information can easily be incorporated into our model using conditions for existence of the package arcs ($b_{u,v}$ in Sec. 5.1.1).

This section presents our computational study. We generate forty problem instances in total, comprised of four sets, each with ten problem instances. The instances we generate are based on the geographical location of hospitals in a metropolitan city in Canada, and publicly available data on the population these hospitals serve. We generate four sets of problem instances, each with a different population distribution among hospitals with differing levels, with each set comprising ten problem instances. Each instance we generate is a list of samples, with each list containing up to 7000 samples. Each sample has the origin hospital, the destination hospital, the time the sample is collected (the start time of the time window), and the deadline (the end time of the time window). The destination hospital is the hospital that has the appropriate level lab. If the hospital where the sample is collected has the appropriate level lab, the origin and destination hospitals are the same, and these samples are not considered in solving our routing problem.

To reduce the size of the input to our model, we pre-process each instance by packing samples which have the same destination as well as a large overlap in their time windows, into a single package. In the extreme case, we will have 140 packages. This, together with the number of hospitals (at most 20), is the size of the input. Although packing procedure is also parametrized, it is relatively simple and trivial. Each instance is run on the Simon Fraser University RCG Colony, a cluster of 64-bit Linux computers (each run is set to use exactly one core of one processor). We specify the details of our computational study below. computational parameters more specifically to the particular set of experiments below.

5.2.1 Comparing Solution Quality across Problem Instances

We use the relative MIP gap, the ratio of the difference between the solution value (obtained by the MIP solver) and either the optimal, or a bound on the optimal, as a measure of the solution quality. We examine its dependence on three parameters: the sparsity of the input graph, the number of vehicles in the fleet, and the discretization time δ used to construct the network. The input graph is either sparse or complete, the number of vehicles ranges from 0 to 25 (in steps of 5), and the discretization time, in minutes, ranges from 5 to 30 (in steps of 5).

We also measure the running time of our model to reach the relative gap of 10% for discretization time steps of 5 minutes and 10 minutes, and 10 vehicles. When δ is 10 minutes (5 minutes), the average time to reach the gap is 1244 seconds (4124 seconds). We set a CPU time limit of 2 hours.

Sparse vs Complete Graph

Our model permits us to specify and exploit the sparsity of the input graph. The underlying input graph may be quite sparse when the geographical area has bridges, highways, and arterial roads. It is clear from Fig. 5.2 that our model requires much less CPU time for sparse graphs. Intuitively, there are more options available in a denser graph. The larger solution space that results slows down the MIP solver.

Missing edges between pairs of nodes may be replaced by ‘edges’ with shortest path distances between corresponding nodes. Adding such missing edges may provide feasible solutions, where none may exist in the sparse graph, due to the fact that we discretize time windows and distances. We evaluate our solutions on the sparse graph in the rest of the paper.

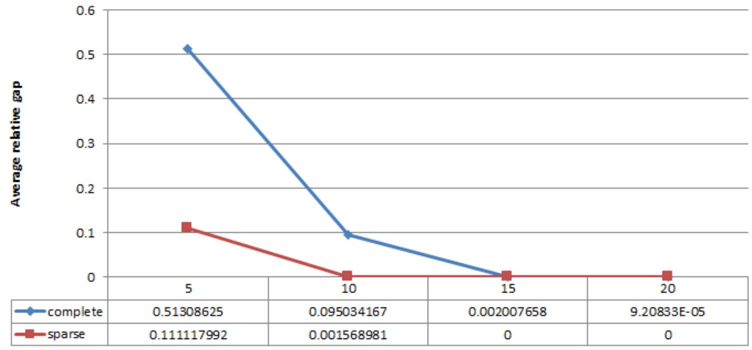

Figure 5.2: Dependence of Average Gap on Discretization for Sparse, and Complete Input Graphs

Number of Vehicles and Discretization Time Steps

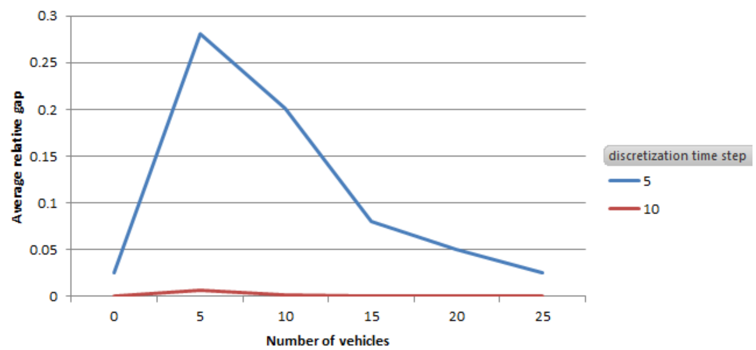

Figure 5.3: Dependence of Average Relative Gap on Number of Vehicles and Discretization Time Steps

Figure 5.3 displays how the relative gap changes with the number of vehicles allowed and the discretization time. We note that solving the problem for the case when the packages have to be delivered by using both vehicles as well as taxis is harder than for the case when the packages have to be delivered either entirely by vehicles or entirely by taxis.

		δ					
		5			10		
		o.v.	gap	taxi calls	o.v.	gap	taxi calls
vehicles	0	22467.38	0.0252	217.175	4519.667	0	46.9444
	5	146686	0.2812	1378.15	1922.944	0.0067	8.3333
	10	128120.1	0.2015	1200	1350.583	0.0023	0.4444
	15	58380.45	0.0805	541.925	1186.167	0.0004	0.3333
	20	38921.08	0.0506	357.425	1135.056	0	0.0556
	25	19329.87	0.0256	171.4359	1129.917	0	0

Table 1: Dependence of Average Objective Function Value, Relative Gap and Number of Taxis on Number of Vehicles and Discretization Time Steps

Table 1 displays how the relative MIP gap, the objective function value, and the number of taxis, depend on the number of vehicles used and the discretization time step. As can be observed, both the objective function value and the number of taxis decrease with the number of vehicles allowed. The flexibility of our model in allowing taxis becomes apparent when few vehicles are present. In these cases, instead of obtaining infeasible solutions, we get solutions with larger objective function value. It turns out that in the real-life scenario that motivated our work, taxis were used whenever it was impossible to transport a sample using a vehicle.

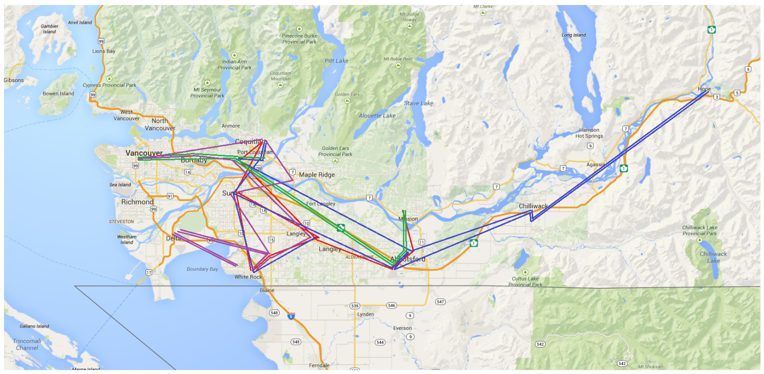

Figure 5.4: Illustrative example of the routes generated using simulated data

5.3 Conclusions and future work

In this work, we address an important problem that arises in the health care industry, that of transporting laboratory samples between hospitals. This problem arises because the laboratories within a hospital may not be equipped to perform the required tests on a sample. We present a mathematical programming formulation of the problem, a solution procedure using CPLEX, and a set of experiments to evaluate the solution procedure. Even though we test our solution procedure on generated data, we believe our solution procedure can be used to solve the real-world problem that motivated this exercise in the first place. The approach outlined in this paper can be applied to solve problems of comparable size that arise in the health care industry.

Future work may include a model-based heuristic that will provide good solutions for larger problem instances. In addition, the geographical area may be partitioned into zones, and the size of the flow network reduced by removing arcs unlikely to be used in an optimal solution. This may permit us to solve much larger instances of the problem.

Chapter 6

Conclusion

In this thesis we are developing the knowledge about few challenging nonlinear assignment problems. Our interest was particularly in computational complexity, solvable special cases, approximations, linearizations and local search algorithms. To build a more complete picture of algorithms performance we have also conducted extensive experimental analysis on real world and generated instances. For several practical applications we have described a detailed process of problem modelling, that heavily relies on results and techniques discussed in this dissertation.

For Bilinear Assignment Problem (BAP) we presented a systematic study of complexity aspects defined on the data set (Q, C, D) and size parameters m and n. BAP generalizes the well known quadratic assignment problem, the three dimensional assignment problem, and the disjoint matching problem. We show that BAP is NP-hard if $m = O(\sqrt[r]{n})$, for some fixed r, but is solvable in polynomial time if $m = O(\frac{\log n}{\log \log n})$. Further, we establish that BAP cannot be approximated within a constant factor unless P=NP even if Q is diagonal; but when the rank of Q is fixed, BAP is observed to admit FPTAS. When the rank of Q is one and C or D is a sum matrix, BAP is shown to be solvable in polynomial time. In contrast, QAP with a diagonal cost matrix is just the linear assignment problem which is solvable in polynomial time. We also provide a characterization of BAP instances equivalent to two linear assignment problems. Same as in the case of QAP, linearizable instances yield a class of polynomially solvable special cases.

Various results leading to performance guarantee of heuristics from the domination analysis point of view are presented. In particular, we showed that a feasible solution with objective function value no worse than that of $(m-1)!(n-1)!$ solutions can be identified efficiently, whereas computing a solution whose objective function value is no worse than that of $m!n! - \lceil \frac{m}{\beta} \rceil ! \lceil \frac{n}{\beta} \rceil !$ solutions is NP-hard for any fixed rational number $\beta > 1$. As a by-product, we have a closed form expression to compute the average of the objective function values of all solutions, but the median of the solution values cannot be identified in polynomial time, unless P=NP.

Moreover, we have presented the first systematic experimental analysis of heuristics for BAP along with some theoretical results on local search algorithms worst case performance. Three classes of neighborhoods - h-exchange, $[h, p]$-exchange and shift based - are introduced. Some of the neighborhoods are of an exponential size but can be searched for an improving solution in polynomial time. Analysis of local optimums in terms of domination properties and relation to average value $\mathcal{A}(Q, C, D)$ are presented.

Several greedy, semi-greedy and rounding construction heuristics are proposed for generating reasonable quality solution quickly. Experimental results show that *RandomXYGreedy* is a good alternative among the approaches. The built-in randomized decision steps make this heuristic valuable for generating starting solutions for improvement algorithms within a multistart framework.

Extensive computational analysis has been carried out on the searches based on described neighborhoods. The experimental results suggest that the very large-scale neighborhood (VLSN) search algorithm - *Alternating Algorithm (AA)*, when used within multistart framework, yields a more balanced heuristic in terms of running time and solution quality. A variable neighborhood search (VNS) algorithm, that strategically uses optimized 2-exchange neighborhood and *AA* neighborhood, produced superior outcomes. However, this came with the downside of a significantly larger computational time.

To assist researchers who pursue experimental study on this versatile optimization model, we have developed several sets of test instances. These instances could serve as a reference point for future performance measures of heuristic as well as exact solution approaches.

Various promising future work areas and challenging open problems, that we have encountered during our work on BAP, could be summarized as follows:

- searching for tighter bounds in complexity and special cases results
- developing FPTAS for more general cases of the problem
- further theoretical and experimental analysis on performance of linearization reformulation techniques
- developing stronger domination analysis results for performance of local search algorithms
- design and analysis of advanced heuristic approaches that are based on presented local search results
- design and analysis of exact algorithms for BAP

On a note of applications, we have presented two real-world problems that have strong connections to nonlinear assignment framework.

The first problem arises in ridesharing systems, and can formally be stated as an assignment of vehicles to groups of passengers that should carpool together. Among others we explore a formulation that relies on assignments combinatorial structure, and is described as the Partition Assignment Problem. We present some complexity results on the problem, including reductions to and from other famous assignment problems. An integer programming formulation is used to obtain solutions to generated instances of various sizes.

This versatile model has not been studied before and therefore presents many areas of open problems such as: analysis of special cases, developing constant factor approximations, further research of domination properties of local search methods, developing heuristics and exact algorithms for the problem.

Another application discussed in this thesis, addressed an important problem that arises in the health care industry, that of transporting laboratory samples between hospitals. This problem arises because the laboratories within a hospital may not be equipped to perform the required tests on a sample. We present a mathematical programming formulation of the problem, a solution procedure using CPLEX, and a set of experiments to evaluate the solution procedure. Even though we test our solution procedure on generated data, we believe our solution procedure can be used to solve the real-world problem that motivated this exercise in the first place. The approach outlined in this paper can be applied to solve problems of comparable size that arise in the health care industry.

Future work may include a model-based heuristic that will provide good solutions for larger problem instances. In addition, the geographical area may be partitioned into zones, and the size of the flow network reduced by removing arcs unlikely to be used in an optimal solution. This may allow to develop approaches to the problem that are better scalable with the size.

Bibliography

[1] Warren P Adams, Monique Guignard, Peter M Hahn, and William L Hightower. A level-2 reformulation–linearization technique bound for the quadratic assignment problem. *European Journal of Operational Research*, 180(3):983–996, 2007.

[2] Ravindra K Ahuja, Özlem Ergun, James B Orlin, and Abraham P Punnen. A survey of very large-scale neighborhood search techniques. *Discrete Applied Mathematics*, 123(1):75–102, 2002.

[3] Ravindra K Ahuja, Özlem Ergun, James B Orlin, and Abraham P Punnen. *Very Large Scale Neighborhood Search: Theory, Algorithms and Applications, Approximation Algorithms and Metaheuristics.* CRC Press, 2007.

[4] Noga Alon, Gregory Gutin, and Michael Krivelevich. Algorithms with large domination ratio. *Journal of Algorithms*, 50(1):118–131, 2004.

[5] M Altman. Bilinear programming. *Bulletin of the polish academy of sciences-series of astronomical and physical mathematic sciences*, 16(9):741, 1968.

[6] Eric Angel and Vassilis Zissimopoulos. On the quality of local search for the quadratic assignment problem. *Discrete Applied Mathematics*, 82(1-3):15–25, 1998.

[7] Kurt M Anstreicher and Nathan W Brixius. Solving quadratic assignment problems using convex quadratic programming relaxations. *Optimization Methods and Software*, 16(1-4):49–68, 2001.

[8] Gautam Appa, D Magos, and Ioannis Mourtos. On multi-index assignment polytopes. *Linear Algebra and its applications*, 416(2-3):224–241, 2006.

[9] Esther M Arkin, Refael Hassin, and Maxim Sviridenko. Approximating the maximum quadratic assignment problem. *Information Processing Letters*, 77(1):13–16, 2001.

[10] Sanjeev Arora, Alan Frieze, and Haim Kaplan. A new rounding procedure for the assignment problem with applications to dense graph arrangement problems. *Mathematical programming*, 92(1):1–36, 2002.

[11] Hans-Jürgen Bandelt, Yves Crama, and Frits CR Spieksma. Approximation algorithms for multi-dimensional assignment problems with decomposable costs. *Discrete Applied Mathematics*, 49(1-3):25–50, 1994.

[12] Xavier Berenguer. A characterization of linear admissible transformations for the m-travelling salesmen problem. *European Journal of Operational Research*, 3(3):232–238, 1979.

[13] Alain Billionnet, Marie-Christine Costa, and Alain Sutter. An efficient algorithm for a task allocation problem. *Journal of the ACM (JACM)*, 39(3):502–518, 1992.

[14] Aurélien Blanchard, Sourour Elloumi, Alain Faye, and Nicolas Wicker. Un algorithme de génération de coupes pour le problème de lâĂŹaffectation quadratique. *INFOR: Information Systems and Operational Research*, 41(1):35–49, 2003.

[15] Shahid H Bokhari. A shortest tree algorithm for optimal assignments across space and time in a distributed processor system. *IEEE transactions on Software Engineering*, (6):583–589, 1981.

[16] Olli Bräysy and Michel Gendreau. Vehicle routing problem with time windows, part i: Route construction and local search algorithms. *Transportation science*, 39(1):104–118, 2005.

[17] Geir Brønmo, Marielle Christiansen, Kjetil Fagerholt, and Bjørn Nygreen. A multi-start local search heuristic for ship schedulingâĂŤa computational study. *Computers & Operations Research*, 34(3):900–917, 2007.

[18] Rainer Burkard, Mauro Dell'Amico, and Silvano Martello. Assignment problems. 2009.

[19] Rainer Burkard, Mauro Dell'Amico, and Silvano Martello. *Assignment problems: revised reprint*. SIAM, 2012.

[20] Rainer E Burkard and Eranda Çela. Heuristics for biquadratic assignment problems and their computational comparison. *European Journal of Operational Research*, 83(2):283–300, 1995.

[21] Rainer E Burkard, Eranda Cela, Günter Rote, and Gerhard J Woeginger. The quadratic assignment problem with a monotone anti-monge and a symmetric toeplitz matrix: Easy and hard cases. *Mathematical Programming*, 82(1):125–158, 1998.

[22] Rainer E Burkard, Bettina Klinz, and Rüdiger Rudolf. Perspectives of monge properties in optimization. *Discrete Applied Mathematics*, 70(2):95–161, 1996.

[23] Rainer E Burkard, Rüdiger Rudolf, and Gerhard J Woeginger. Three-dimensional axial assignment problems with decomposable cost coefficients. *Discrete Applied Mathematics*, 65(1-3):123–139, 1996.

[24] Avishai Ceder. Designing public transport networks and routes. In *Advanced Modeling for Transit Operations and Service Planning*, pages 59–91. Emerald Group Publishing Limited, 2002.

[25] Eranda Cela. *The quadratic assignment problem: theory and algorithms*, volume 1. Springer Science & Business Media, 2013.

[26] Eranda Çela, Rainer E Burkard, and Bettina Klinz. On the biquadratic assignment problem. In *Quadratic Assignment and Related Problems: DIMACS Workshop, May 20-21, 1993*, volume 16, pages 117–146. American Mathematical Soc., 1994.

[27] Eranda Çela, Vladimir G Deineko, and Gerhard J Woeginger. Linearizable special cases of the qap. *Journal of Combinatorial optimization*, 31(3):1269–1279, 2016.

[28] Dilip Chhajed and Timothy J Lowe. m-median and m-center problems with mutual communication: Solvable special cases. *Operations Research*, 40(1-supplement-1):S56–S66, 1992.

[29] Marielle Christiansen, Kjetil Fagerholt, and David Ronen. Ship routing and scheduling: Status and perspectives. *Transportation science*, 38(1):1–18, 2004.

[30] Nicos Christofides and Enrique Benavent. An exact algorithm for the quadratic assignment problem on a tree. *Operations Research*, 37(5):760–768, 1989.

[31] Jean-François Cordeau, Gilbert Laporte, Martin WP Savelsbergh, and Daniele Vigo. Vehicle routing. *Handbooks in operations research and management science*, 14:367–428, 2007.

[32] Cristián E Cortés, Martín Matamala, and Claudio Contardo. The pickup and delivery problem with transfers: Formulation and a branch-and-cut solution method. *European Journal of Operational Research*, 200(3):711–724, 2010.

[33] Yves Crama and Frits CR Spieksma. Approximation algorithms for three-dimensional assignment problems with triangle inequalities. *European Journal of Operational Research*, 60(3):273–279, 1992.

[34] Ante Ćustić and Bettina Klinz. The constant objective value property for multidimensional assignment problems. *Discrete Optimization*, 19:23–35, 2016.

[35] Ante Ćustić, Bettina Klinz, and Gerhard J Woeginger. Geometric versions of the three-dimensional assignment problem under general norms. *Discrete Optimization*, 18:38–55, 2015.

[36] Ante Ćustić and Abraham P Punnen. A characterization of linearizable instances of the quadratic minimum spanning tree problem. *arXiv preprint arXiv:1510.02197*, 2015.

[37] Ante Ćustić and Abraham P Punnen. Average value of solutions of the bipartite quadratic assignment problem and linkages to domination analysis. *Operations Research Letters*, 45(3):232–237, 2017.

[38] Ante Ćustić, Vladyslav Sokol, Abraham P Punnen, and Binay Bhattacharya. The bilinear assignment problem: complexity and polynomially solvable special cases. *Mathematical Programming*, 166(1-2):185–205, 2017.

[39] Vladimir G De&ıbreve, Gerhard J Woeginger, et al. A study of exponential neighborhoods for the travelling salesman problem and for the quadratic assignment problem. *Mathematical programming*, 87(3):519–542, 2000.

[40] Martin Desrochers, Jacques Desrosiers, and Marius Solomon. A new optimization algorithm for the vehicle routing problem with time windows. *Operations research*, 40(2):342–354, 1992.

[41] Wolfgang Domschke. Schedule synchronization for public transit networks. *Operations-Research-Spektrum*, 11(1):17–24, 1989.

[42] Wolfgang Domschke, Peter Forst, and Stefan Voß. Tabu search techniques for the quadratic semi-assignment problem. In *New directions for operations research in manufacturing*, pages 389–405. Springer, 1992.

[43] Zvi Drezner, Peter M Hahn, and Éeric D Taillard. Recent advances for the quadratic assignment problem with special emphasis on instances that are difficult for meta-heuristic methods. *Annals of Operations research*, 139(1):65–94, 2005.

[44] Thomas A Feo and Mauricio GC Resende. A probabilistic heuristic for a computationally difficult set covering problem. *Operations Research Letters*, 8(2):67–71, 1989.

[45] Dmitri G Fon-Der-Flaass. Arrays of distinct representativesâĂŤa very simple np-complete problem. *Discrete Mathematics*, 171(1-3):295–298, 1997.

[46] Alan M Frieze. A bilinear programming formulation of the 3-dimensional assignment problem. *Mathematical Programming*, 7(1):376–379, 1974.

[47] Alan M Frieze. Complexity of a 3-dimensional assignment problem. *European Journal of Operational Research*, 13(2):161–164, 1983.

[48] Alan M Frieze, J Yadegar, S El-Horbaty, and D Parkinson. Algorithms for assignment problems on an array processor. *Parallel Computing*, 11(2):151–162, 1989.

[49] Alan M Frieze and Joseph Yadegar. On the quadratic assignment problem. *Discrete applied mathematics*, 5(1):89–98, 1983.

[50] Michael R Garey and David S Johnson. *Computers and intractability*, volume 29. wh freeman New York, 2002.

[51] E Kh Gimadi and N Korkishko. On some modifications of the three index planar assignment problem. In *Discrete optimization methods in production and logistics. The second int. workshop, Omsk*, pages 161–165, 2004.

[52] Edward Kh Gimadi and Natalie M Kairan. Multi-index assignment problem: an asymptotically optimal approach. In *Emerging Technologies and Factory Automation, 2001. Proceedings. 2001 8th IEEE International Conference on*, volume 2, pages 707–709. IEEE, 2001.

[53] Fred Glover and Abraham P Punnen. The travelling salesman problem: new solvable cases and linkages with the development of approximation algorithms. *Journal of the Operational Research Society*, 48(5):502–510, 1997.

[54] Fred Glover, Tao Ye, Abraham P Punnen, and Gary Kochenberger. Integrating tabu search and vlsn search to develop enhanced algorithms: A case study using bipartite boolean quadratic programs. *European Journal of Operational Research*, 241(3):697–707, 2015.

[55] Bruce L Golden, Subramanian Raghavan, and Edward A Wasil. *The vehicle routing problem: latest advances and new challenges*, volume 43. Springer Science & Business Media, 2008.

[56] Dries Goossens, Sergey Polyakovskiy, Frits CR Spieksma, and Gerhard J Woeginger. Between a rock and a hard place: the two-to-one assignment problem. *Mathematical Methods of Operations Research*, 76(2):223–237, 2012.

[57] Ram Gopalan and Kalyan T Talluri. Mathematical models in airline schedule planning: A survey. *Annals of Operations Research*, 76:155–185, 1998.

[58] Alex Grasas, Helena Ramalhinho, Luciana S Pessoa, Mauricio GC Resende, Imma Caballé, and Nuria Barba. On the improvement of blood sample collection at clinical laboratories. *BMC health services research*, 14(1):12, 2014.

[59] Glenn William Graves and Andrew B Whinston. An algorithm for the quadratic assignment problem. *Management Science*, 16(7):453–471, 1970.

[60] Harold Greenberg. A quadratic assignment problem without column constraints. *Naval Research Logistics (NRL)*, 16(3):417–421, 1969.

[61] Lov K Grover. Local search and the local structure of np-complete problems. *Operations Research Letters*, 12(4):235–243, 1992.

[62] Don A Grundel, Pavlo A Krokhmal, Carlos AS Oliveira, and Panos M Pardalos. On the number of local minima for the multidimensional assignment problem. *Journal of Combinatorial Optimization*, 13(1):1–18, 2007.

[63] Gregory Gutin, Tommy Jensen, and Anders Yeo. Domination analysis for minimum multiprocessor scheduling. *Discrete applied mathematics*, 154(18):2613–2619, 2006.

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

[65] Gregory Gutin, Alek Vainshtein, and Anders Yeo. Domination analysis of combinatorial optimization problems. *Discrete Applied Mathematics*, 129(2):513–520, 2003.

[66] Gregory Gutin and Anders Yeo. Tsp tour domination and hamilton cycle decompositions of regular digraphs. *Operations Research Letters*, 28(3):107–111, 2001.

[67] Gregory Gutin and Anders Yeo. Polynomial approximation algorithms for the tsp and the qap with a factorial domination number. *Discrete Applied Mathematics*, 119(1):107–116, 2002.

[68] Peter Hahn, Thomas Grant, and Nat Hall. A branch-and-bound algorithm for the quadratic assignment problem based on the hungarian method. *European Journal of Operational Research*, 108(3):629–640, 1998.

[69] Pierre Hansen, Nenad Mladenović, Raca Todosijević, and Saïd Hanafi. Variable neighborhood search: basics and variants. *EURO Journal on Computational Optimization*, pages 1–32, 2016.

[70] J Pirie Hart and Andrew W Shogan. Semi-greedy heuristics: An empirical study. *Operations Research Letters*, 6(3):107–114, 1987.

[71] Refael Hassin and Samir Khuller. z-approximations. *Journal of Algorithms*, 41(2):429–442, 2001.

[72] Dirk Hausmann, Bernhard Korte, and TA Jenkyns. Worst case analysis of greedy type algorithms for independence systems. In *Combinatorial Optimization*, pages 120–131. Springer, 1980.

[73] Cor A. J. Hurkens and Alexander Schrijver. 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 Journal on Discrete Mathematics*, 2(1):68–72, 1989.

[74] Santosh N Kabadi and Abraham P Punnen. An o (n 4) algorithm for the qap linearization problem. *Mathematics of Operations Research*, 36(4):754–761, 2011.

[75] Brian Kallehauge, Jesper Larsen, Oli BG Madsen, and Marius M Solomon. Vehicle routing problem with time windows. In *Column generation*, pages 67–98. Springer, 2005.

[76] Daniel Karapetyan and Abraham P Punnen. Heuristic algorithms for the bipartite unconstrained 0-1 quadratic programming problem. *arXiv preprint arXiv:1210.3684*, 2012.

[77] Daniel Karapetyan and Abraham P Punnen. A reduced integer programming model for the ferry scheduling problem. *Public Transport*, 4(3):151–163, 2013.

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

[79] L Kaufman and Fernand Broeckx. An algorithm for the quadratic assignment problem using bender's decomposition. *European Journal of Operational Research*, 2(3):207–211, 1978.

[80] Arman Kaveh. Algorithms and theoretical topics on selected combinatorial optimization problems. Master's thesis, Science: Department of Mathematics, 2010.

[81] Bettina Klinz and Gerhard J Woeginger. A new efficiently solvable special case of the three-dimensional axial bottleneck assignment problem. In *Combinatorics and Computer Science*, pages 150–162. Springer, 1996.

[82] Niklas Kohl, Jacques Desrosiers, Oli BG Madsen, Marius M Solomon, and Francois Soumis. 2-path cuts for the vehicle routing problem with time windows. *Transportation Science*, 33(1):101–116, 1999.

[83] Niklas Kohl and Oli BG Madsen. An optimization algorithm for the vehicle routing problem with time windows based on lagrangian relaxation. *Operations research*, 45(3):395–406, 1997.

[84] AE Koller and SD Noble. Domination analysis of greedy heuristics for the frequency assignment problem. *Discrete Mathematics*, 275(1):331–338, 2004.

[85] Hiroshi Konno. Maximization of a convex quadratic function under linear constraints. *Mathematical programming*, 11(1):117–127, 1976.

[86] Hiroshi Konno. Maximizing a convex quadratic function over a hypercube. *Journal of the Operations Research Society of Japan*, 23(2):171–189, 1980.

[87] Tjalling C Koopmans and Martin Beckmann. Assignment problems and the location of economic activities. *Econometrica: journal of the Econometric Society*, pages 53–76, 1957.

[88] VM Kravtsov. Polynomial algorithms for finding the asymptotically optimum plan of the multiindex axial assignment problem. *Cybernetics and Systems Analysis*, 41(6):940–944, 2005.

[89] Daniela Kühn and Deryk Osthus. Hamilton decompositions of regular expanders: a proof of kellyâĂŹs conjecture for large tournaments. *Advances in Mathematics*, 237:62–146, 2013.

[90] Harold W Kuhn. The hungarian method for the assignment problem. *Naval Research Logistics Quarterly*, 2(1-2):83–97, 1955.

[91] Jeroen Kuipers. *On the LP-relaxation of Multi-dimensional Assignment Problems with Applications to Assignment Games.* University of Limburg, 1990.

[92] Gilbert Laporte. The vehicle routing problem: An overview of exact and approximate algorithms. *European journal of operational research*, 59(3):345–358, 1992.

[93] Gilbert Laporte, Michel Gendreau, J-Y Potvin, and Frédéric Semet. Classical and modern heuristics for the vehicle routing problem. *International transactions in operational research*, 7(4-5):285–300, 2000.

[94] Eugene L Lawler. The quadratic assignment problem. *Management Science*, 9(4):586–599, 1963.

[95] Eliane Maria Loiola, Nair Maria Maia de Abreu, Paulo Oswaldo Boaventura-Netto, Peter Hahn, and Tania Querido. A survey for the quadratic assignment problem. *European journal of operational research*, 176(2):657–690, 2007.

[96] Federico Malucelli and Daniele Pretolani. Lower bounds for the quadratic semi-assignment problem. *European Journal of Operational Research*, 83(2):365–375, 1995.

[97] Snežana Mitrović-Minić and Gilbert Laporte. The pickup and delivery problem with time windows and transshipment. *INFOR: Information Systems and Operational Research*, 44(3):217–227, 2006.

[98] Snežana Mitrović-Minić and Abraham P Punnen. Local search intensified: Very large-scale variable neighborhood search for the multi-resource generalized assignment problem. *Discrete Optimization*, 6(4):370–377, 2009.

[99] Shashi Mittal and Andreas S Schulz. An fptas for optimizing a class of low-rank functions over a polytope. *Mathematical Programming*, 141(1-2):103–120, 2013.

[100] Kowtha A Murthy, Yong Li, and Panos M Pardalos. A local search algorithm for the quadratic assignment problem. *Informatica*, 3(4):524–538, 1992.

[101] Christos H Papadimitriou and David Wolfe. The complexity of facets resolved. *Journal of Computer and System Sciences*, 37(1):2–13, 1988.

[102] Panos M Pardalos, KG Ramakrishnan, Mauricio GC Resende, and Yong Li. Implementation of a variance reduction-based lower bound in a branch-and-bound algorithm for the quadratic assignment problem. *SIAM Journal on Optimization*, 7(1):280–294, 1997.

[103] Panos M Pardalos, Henry Wolkowicz, et al. *Quadratic Assignment and Related Problems: DIMACS Workshop, May 20-21, 1993*, volume 16. American Mathematical Soc., 1994.

[104] William P Pierskalla. Letter to the editor-the multidimensional assignment problem. *Operations Research*, 16(2):422–431, 1968.

[105] Abraham Punnen and Santosh Kabadi. Domination analysis of some heuristics for the traveling salesman problem. *Discrete Applied Mathematics*, 119(1):117–128, 2002.

[106] Abraham Punnen, Francois Margot, and Santosh Kabadi. Tsp heuristics: domination analysis and complexity. *Algorithmica*, 35(2):111–127, 2003.

[107] Abraham P Punnen and Santosh N Kabadi. A linear time algorithm for the koopmans–beckmann qap linearization and related problems. *Discrete Optimization*, 10(3):200–209, 2013.

[108] Abraham P Punnen, Piyashat Sripratak, and Daniel Karapetyan. Average value of solutions for the bipartite boolean quadratic programs and rounding algorithms. *Theoretical Computer Science*, 565:77–89, 2015.

[109] Abraham P Punnen and Yang Wang. The bipartite quadratic assignment problem and extensions. *European Journal of Operational Research*, 250(3):715–725, 2016.

[110] Maurice Queyranne. Performance ratio of polynomial heuristics for triangle inequality quadratic assignment problems. *Operations Research Letters*, 4(5):231–234, 1986.

[111] Arash Rafiey, Vladyslav Sokol, Ramesh Krishnamurti, Snezana Mitrovic-Minic, Abraham P Punnen, and Krishna Teja Malladi. A network model for the hospital routing problem. In *ICORES*, pages 353–358, 2015.

[112] Abdur Rais and Ana Viana. Operations research in healthcare: a survey. *International transactions in operational research*, 18(1):1–31, 2011.

[113] F Rendl. The quadratic assignment problem. *Facility location: applications and theory. Springer, Berlin*, pages 439–457, 2002.

[114] Franz Rendl. On the complexity of decomposing matrices arising in satellite communication. *Operations Research Letters*, 4(1):5–8, 1985.

[115] VI Rublineckii. Estimates of the accuracy of procedures in the traveling salesman problem. *Numerical Mathematics and Computer Technology*, 4:18–23, 1973.

[116] Sartaj Sahni and Teofilo Gonzalez. P-complete approximation problems. *Journal of the ACM (JACM)*, 23(3):555–565, 1976.

[117] VI Sarvanov. The mean value of the functional in sampling problems, vestsi akademii navuk bssr. *Seryya Fizika-Matematychnykh Navuk*, 139:51–54, 1978.

[118] VI Sarvanov and NN Doroshko. The approximate solution of the traveling salesman problem by a local algorithm that searches neighborhoods of exponential cardinality in quadratic time. *Software: Algorithms and Programs*, 31:8–11, 1981.

[119] VI Sarvanov and NN Doroshko. Approximate solution of the traveling salesman problem by a local algorithm with scanning neighbourhoods of factorial cardinality in cubic time. *Software: Algorithms and Programs*, 31:11–13, 1981.

[120] Martin WP Savelsbergh and Marc Sol. The general pickup and delivery problem. *Transportation science*, 29(1):17–29, 1995.

[121] Vladyslav Sokol, Ante Ćustić, Abraham P Punnen, and Binay Bhattacharya. Bilinear assignment problem: Large neighborhoods and experimental analysis of algorithms. *arXiv preprint arXiv:1707.07057*, 2017.

[122] Frits CR Spieksma. Multi index assignment problems: complexity, approximation, applications. In *Nonlinear Assignment Problems*, pages 1–12. Springer, 2000.

[123] Frits CR Spieksma and Gerhard J Woeginger. Geometric three-dimensional assignment problems. *European Journal of Operational Research*, 91(3):611–618, 1996.

[124] Leon Steinberg. The backboard wiring problem: A placement algorithm. *Siam Review*, 3(1):37–50, 1961.

[125] Sergei Pavlovich Tarasov. Properties of the trajectories of the appointments problem and the travelling-salesman problem. *USSR Computational Mathematics and Mathematical Physics*, 21(1):167–174, 1981.

[126] Abdolhamid Torki, Yatsutoshi Yajima, and Takao Enkawa. A low-rank bilinear programming approach for sub-optimal solution of the quadratic assignment problem. *European Journal of Operational Research*, 94(2):384–391, 1996.

[127] Paolo Toth and Daniele Vigo. The vehicle routing problem, ser. siam monographs on discrete mathematics and applications. *Society for Industrial and Applied Mathematics*, 2002.

[128] Louis Y Tsui and Chia-Hao Chang. A microcomputer based decision support tool for assigning dock doors in freight yards. *Computers & Industrial Engineering*, 19(1-4):309–312, 1990.

[129] Louis Y Tsui and Chia-Hao Chang. An optimal solution to a dock door assignment problem. *Computers & Industrial Engineering*, 23(1-4):283–286, 1992.

[130] Yochai Twitto. Dominance guarantees for above-average solutions. *Discrete Optimization*, 5(3):563–568, 2008.

[131] VG Vizing. Values of the target functional in a priority problem that are majorized by the mean value. *Kibernetika, Kiev*, 5:76–78, 1973.

[132] Stefan Voss. Network design formulations in schedule synchronization. In *Computer-Aided Transit Scheduling*, pages 137–152. Springer, 1992.

[133] Eitan Zemel. Measuring the quality of approximate solutions to zero-one programming problems. *Mathematics of operations research*, 6(3):319–332, 1981.

[134] Karel Zikan. Track initialization in the multiple-object tracking problem. Technical report, Stanford University Systems Optimization Lab, 1988.