FACULTEIT ECONOMIE EN
BEDRIJFSWETENSCHAPPEN



KU LEUVEN

COMBINATORIAL OPTIMIZATION PROBLEMS WITH CONFLICTS OR VECTORS

Proefschrift voorgedragen tot
het behalen van de graad van
Doctor in de Toegepaste
Economische Wetenschappen

door

Annette M.C. Ficker

NUMMER 606 2018

DOCTORAL COMMITTEE

Advisor:	
Prof. dr. Frits C.R. Spieksma	*KU Leuven & TU Eindhoven*
Co-Advisor:	
Prof. dr. Roel Leus	*KU Leuven*
Members:	
Prof. dr. Erik Demeulemeester	*KU Leuven*
Prof. dr. Dries R. Goossens	*UGent*
Prof. dr. Marc Uetz	*TU Twente*
Prof. dr. Gerhard J. Woeginger	*RWTH Aachen*

Daar de proefschriften in de reeks van de Faculteit Economie en Bedrijfswetenschappen het persoonlijk werk zijn van hun auteurs, zijn alleen deze laatsten daarvoor verantwoordelijk.

Amazing Grace

ACKNOWLEDGMENTS

> If I have been able to see further,
> it was only because I stood on the
> shoulders of giants.
>
> *Isaac Newton*

Many people have helped and supported me, directly or indirectly. It is therefore my honour and pleasure to give thanks to so many wonderful people. In fact, so many people have supported me that it is nearly impossible to mention everyone. I will make an attempt anyway, but would like to offer my sincere apologies to anyone I may have forgotten.

First of all I would like to thank my PhD advisor Frits Spieksma. Thank you for this incredible adventure. I've often jokingly told others that the title of the thesis would be "anything I found interesting", which is in a way a more fitting title. You have given me the freedom to explore anything I found interesting. You were always there to give advice, if not in person then via email, and often you have helped me to take an idea to a higher level.

Frits, I know you as an optimistic, encouraging and gentle person and I want to thank you for your support. You created a research environment for me which made everything possible.

Naturally, I would like to thank my co-advisor Roel Leus next, your challenging and interesting questions have broadened my view on Operations Research. When Frits was not around, I knew I could come to you for advice and you were always willing to listen and help me out. Roel, I admire you appreciation for a wide range of research topics, you value each of them in their own right.

Special thanks to Gerhard Woeginger, twice you greatly influenced my future. The first time, when you gave a masterclass on stable and perfect matchings to high school students. That day my enthusiasm for combinatorial optimization was born and you encouraged me to study mathematics. The second time, when you told me about a PhD position in Leuven and recommended me to apply.

I am thankful for all you taught me and sincerely enjoyed our complexity discussions.

I also would like to thank the rest of the doctoral committee, Erik Demeulemeester, Dries Goossens and Marc Uetz. Thank you for reading my thesis and all the helpful comments you have given me. They have greatly improved my thesis.

Moving to a new place is always exciting and challenging, moving to another country even more. It is also a great opportunity to get to know new people and make new friends, and so I did.

First I would like to thank my academic brothers Bart, Bart and Ward. Not only did we have interesting discussions, we also had a lot of fun together. I especially enjoyed the boardgame evenings and our extended stay in Glasgow after the conference. Bart, my favorite go-to person for a good complexity discussion, thank you for the advice you gave me, which as (jokingly) promised, I have added at the end of the acknowledgments. Bart, my personal Flemish teacher, with you there is no such thing as a boring day in the office. Ward, our Latex and C++ hero, I do not know anyone else who decrypts error messages with such joy as you do.

Of course I had many more great colleagues; from the 'fifth floor' Kris, Daniel, Guopeng, Fan, Salim, Ben and Erik, from the 'fourth floor' Mieke, Ann, Jorne, Dennis, Joeri, Michael, Hamed, Morteza, Stef, Raïsa, Carla, Valeria, Joren I & II, Sarah, Sebastian, Heletj'e, Laurens, Kim and Sha. Thank you all for the fun lunch breaks, movie nights, cake days, the awesome days of awesomeness and many more. I have many great memories from my time in Leuven thanks to you.

I also met some amazing people outside academia. I would like to thank everyone from Vineyard Brussels, you have supported me in so many ways and have helped me feel at home in Belgium. Special thanks to Mark & Gabriel, Thandile, Elisabeth, John and Thomas, you are like family to me.

Having moved to another country I inevitably left some people behind in the Netherlands. There are a few people that I want to thank explicitly.

First some of my fellow mathematicians whom I met in Eindhoven. Tim and Jorn, I thank you both for the great (and at times loud) discussions about our homework, and all the fun activities we did together

when we were not studying. Maria, thank you for everything you taught me, I am grateful for all the advice you gave me during my PhD.

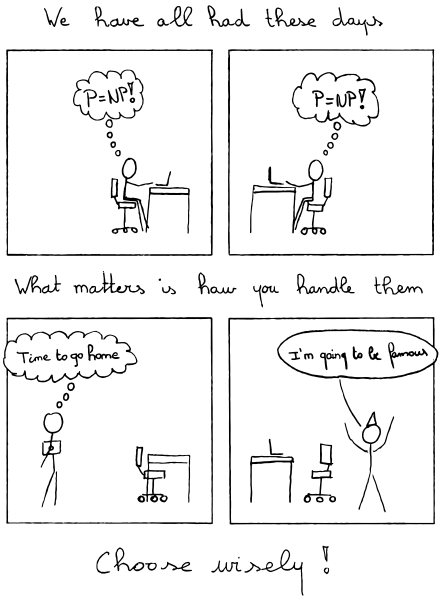
Next my dear friends Lianne, Ruben, Elmer and Margreeth. Each of you always knew how to cheer me up, knew exactly what I needed to hear. And you would always find time to do something fun together, thank you.

Last, but not least, I would like to thank my family.

Als eerste mijn ouders, bedankt voor alles wat jullie voor mij gedaan hebben, zonder jullie had ik dit niet kunnen doen. Gerben en Kim, bedankt ook voor jullie geduld, hopelijk kunnen we nu eindelijk eens samen een bordspel spelen. Mijn schoonfamilie zou ik graag willen bedanken om mij zo welkom te heten.

Als laatste, maar niet de minste, Reint. Met jou trouwen is een van de beter keuzes die ik gemaakt heb in mijn leven. Bedankt voor alles wat je voor me gedaan hebt.

Annette Ficker
Leuven, June 13, 2018.



CONTENTS

1 A FRIENDLY INTRODUCTION 1
1.1 Combinatorial Optimization Problems . . . 1
1.2 Combinatorial Optimization with Conflicts or Vectors . . . 2
1.2.1 Balanced Optimization with Vector Costs . . . 2
1.2.2 The Transportation Problem with Conflicts . . . 4
1.2.3 Partitioning Vectors into Quadruples . . . 5

2 ROBUST BALANCED OPTIMIZATION 7
2.1 Introduction . . . 7
2.1.1 Related Literature . . . 9
2.1.2 Our Results . . . 12
2.2 The Framework . . . 13
2.3 Algorithms for Robust Balanced Optimization Problems . 15
2.3.1 Fixed Dimension . . . 15
2.3.2 Approximation Algorithms . . . 17
2.4 The Complexity of Robust Balanced Optimization . . . 19
2.4.1 q-Uniform Set Systems . . . 20
2.4.2 Linear Assignment . . . 22
2.4.3 Spanning Tree . . . 24
2.4.4 s, t-Cut . . . 26
2.4.5 Connecting Path . . . 28
2.4.6 Horn-SAT . . . 31
2.4.7 2SAT . . . 33
2.5 A Special Case: Linear Assignment with Sum Costs . . . 34
2.6 Computational Experiments for Linear Assignment . . . 39
2.6.1 MIP Formulation . . . 39
2.6.2 Construction of Data-sets . . . 40
2.6.3 Details of Implementation . . . 43
2.6.4 Results . . . 44
2.7 Conclusion . . . 51

3 THE TRANSPORTATION PROBLEM WITH CONFLICTS 53
3.1 Introduction . . . 53
3.1.1 Problem Statement . . . 53
3.1.2 Terminology and Notation . . . 55

3.1.3 Related Literature . . . 57
3.2 A Preview . . . 61
3.2.1 Our Results . . . 62
3.3 Complexity Results Depending on Conflict Graphs . . . 64
3.3.1 Matching or Simple Path . . . 64
3.3.2 Split Graph . . . 70
3.3.3 Complete Bipartite and $s_i = 1$, $d_j = 2$ . . . 73
3.4 Approximation for Max-TPC . . . 77
3.4.1 Max-TPC Meets Separable Assignment Problem . . . 78
3.5 Conclusion . . . 82

4 PARTITIONING VECTORS INTO QUADRUPLES 85
4.1 Introduction . . . 85
4.1.1 Related Literature . . . 87
4.2 Preliminaries . . . 89
4.2.1 Special Cases of Problem PQ . . . 89
4.2.2 The Complexity of Problem PQ . . . 91
4.2.3 About Algorithm A: Notation and Properties . . . 92
4.2.4 Our Results . . . 95
4.3 Upper Bound Approximation Guarantee . . . 96
4.3.1 PQ . . . 96
4.3.2 PQ(#1 = 2) . . . 107
4.3.3 PQ(#1 = 2, distinct) . . . 116
4.3.4 PQ(#1 = 2, distinct, connected) . . . 117
4.4 Lower Bound Approximation Guarantee . . . 121
4.4.1 PQ(#1 ∈ {1, 2}) . . . 121
4.4.2 PQ(#1 = 2, connected) . . . 122
4.4.3 PQ(#1 = 2, distinct, connected) . . . 124
4.5 Bad instances for a natural greedy algorithm . . . 125
4.5.1 PQ . . . 126
4.5.2 PQ(#1 = 2, distinct, connected) . . . 126
4.6 Conclusion . . . 127

List of Figures 129

List of Tables 131

Bibliography 133

Dissertations from the Faculty of Economics and Business 141

1

A FRIENDLY INTRODUCTION

> The problem is not the problem,
> the problem is your attitude about
> the problem.
>
> *Jack Sparrow*

1.1 COMBINATORIAL OPTIMIZATION PROBLEMS

Every day, algorithms are at work to solve combinatorial optimization problems to help us. There are, for example, algorithms that help a parcel delivery company to decide how to deliver all parcels at minimum cost. And when we use a navigation system there is an algorithm at work determining the shortest path to our destination. These are just two examples, and there are many, many more.

A problem is called a combinatorial optimization problem when the set of possible solutions is either finite or countably infinite, and the objective is either to minimize or to maximize. An instance of a problem is defined by a given set of possible solutions (implicitly defined) and for any of these solutions we can determine its value. In our navigation system example, every route from our current location to our destination would be a possible solution. One can determine the total travel distance of each route and the goal is to find a route with the shortest total travel distance.

One way to find an optimal solution to a combinatorial optimization problem could be to simply enumerate all possible solutions and pick the best one. However, often there are too many possible solutions to

compare in a reasonable amount of time. Therefore it is important to find efficient algorithms to solve combinatorial optimization problems.

Some combinatorial optimization problems are in problem class P; meaning that there is an algorithm with a running time bounded by a polynomial in the size of the input, that solves each instance of the problem to optimality. However, many combinatorial optimization problems are NP-hard. Roughly speaking, this means that (unless P=NP) there are no polynomial-time algorithms that solve every instance to optimality. When we prove that a problem is solvable in polynomial time, or is NP-hard, we say that we have established the *complexity* of the problem. We refer to Garey and Johnson (1979) for an extensive study on this topic.

If we cannot find an optimal solution in polynomial time, then perhaps we can find a "good enough" solution in polynomial time. Ideally, we have some idea how much the value of this "good enough" solution differs (at most) from the value of an optimal solution. One such group of algorithms is called α-approximation algorithms; these algorithms guarantee that for every possible instance the value of the solution found is within a factor α from the value of an optimal solution. It follows that $\alpha \geqslant 1$ for minimization problems and $\alpha \leqslant 1$ for maximization problems. For more information on approximation algorithms we refer to Vazirani (2001) and Williamson and Shmoys (2011).

In this thesis we introduce new combinatorial optimization problems, establish their complexity and present approximation algorithms to solve them.

1.2 COMBINATORIAL OPTIMIZATION WITH CONFLICTS OR VECTORS

1.2.1 *Balanced Optimization with Vector Costs*

Consider a manufacturer, who produces products that consist of two parts that need to be put together. The manufacturer first produces sev-

eral copies of the first part and then equally many copies of the second part. Ideally, all copies are identical to its design, but in the production step minor deviations from the original design arise. Suppose that for any combination of a first and second part there is a number expressing the deviation of the end-product compared to their design. The lower this number, the more the end-product equals its design. The problem is to decide which first part should be connected to which second part.

One could use a classical Assignment Problem to find an assignment of first parts to second parts with total minimum deviation. This, however, might give you products of diverse quality. The goal of a *Balanced Optimization problem* is to find a solution of uniform quality. In our example it would return an assignment in which the difference between the largest and smallest deviation of the end-products is minimized.

In Chapter 2, we introduce a family of Balanced Optimization Problems with vector costs. These problems have different practical applications. In particular, the Balanced Assignment Problem with vector costs can be used to model a production planning problem where the aim is to find a set of products of uniform quality. Indeed, Kamura and Nakamori (2014) sketch an industrial problem in the manufacturing of glass lenses that gives rise to a special case of Balanced Assignment Problem with vector costs. In addition, Balanced Optimization Problems with vector costs can be interpreted as Robust Balanced Optimization. In Robust Optimization we are not sure about the actual costs and hence we are given a set of scenario's. The goal is to find a solution which minimizes the worst possible outcome over all scenarios.

We investigate in Chapter 2 the complexity and approximability of Robust Balanced Optimization Problems in a fairly general setting. We identify a large family of problems that admit a 2-approximation in polynomial time, and we show that for many problems in this family this approximation factor 2 is best-possible (unless P=NP). We pay special attention to the balanced assignment problem with vector costs and show that this problem is NP-hard even in the highly restricted case of sum costs. We conclude with computational experiments for this problem.

1.2.2 *The Transportation Problem with Conflicts*

Consider the classical Transportation Problem, in which we are given suppliers, each having a supply, and locations, each having a demand. For all possible pairs consisting of a supplier and a location we are given a unit transportation cost. The goal is to fulfill the demand with minimum cost. This problem is well-known and solvable in polynomial time, see Ahuja et al. (1993).

Many situations in practice have, as a base, this transportation problem. However, additional properties are often present. To illustrate this, consider a setting described by Vancroonenburg et al. (2014), where patients (suppliers) have to be allocated to hospital rooms (locations), with the additional constraint that each room should only contain patients of the same gender. We call a pair of patients with different gender a *forbidden pair*, and further, we call the set of forbidden pairs the *conflict set*. This example gives rise to the so-called Red-Blue Transportation Problem.

Another example, see Cao (1992), comes from storage management where containers (suppliers) need to be placed in rows of a storage yard (locations), such that the cost of operations (search, load) is minimized. However, some containers are not allowed to be placed in the same row, due to their content or size. Again, two containers that cannot be placed in the same row are called a forbidden pair, and the set of forbidden pairs for a particular row forms its conflict set. The resulting situation is slightly more general than the Red-Blue Transportation Problem and is called the Transportation Problem with Exclusionary Side Constraints.

Our last example comes from Chen et al. (2016), where companies (suppliers) want to promote their products to potential customers (locations). On the one hand, a customer wants to limit the number of promotions received from similar companies, inducing forbidden pairs of companies for each customer. On the other hand, companies want to geographically spread their promotion and therefore limit the number of promotions to customers living close to each other, inducing forbid-

den pairs of customers. Chen et al. (2016) call this problem the Conflict-aware weighted Bipartite b-matching problem.

In Chapter 3 we introduce the so-called Transportation Problem with Conflicts, which generalizes all the examples mentioned above, by extending the input of the classical transportation problem with a conflict set for each supplier and each location. Since conflict sets consist of forbidden pairs we can model a conflict set with a so-called conflict graph. A conflict graph belonging to a specific graph class indicates a specific structure of the conflict set. In this chapter we study the complexity and the approximability of the Transportation Problem with Conflicts depending upon different classes of conflict graphs.

1.2.3 *Partitioning Vectors into Quadruples*

Consider a packager, who is given a set of items, for example drinking glasses, that need to be packaged 4 items in a box. In theory these items are identical, but in practice each of these items differs in terms of quality. Hence, it matters which 4 items are packed together. Suppose now that for each item we are given a vector, expressing several measurements of potential weaknesses. The lower such number, the lesser this weakness is present. The problem is to find a partition of items into boxes of 4 items at minimum total weakness. The total weakness is determined by the sum of the weaknesses of each box. And the weakness of each box is determined by the sum of the component-wise maximum of the four vectors representing the potential weaknesses of the four items packed in that box.

This objective function is also used in the production of so-called waferstacks, which are used in the production process for computer chips. One single wafer consists of many dies, each of which can be in two states: either functioning, or malfunctioning. A die in a waferstack is malfunctioning as soon as that die is malfunctioning in at least one wafer of that stack. The goal is to partition the set of wafers into wafer-

stacks such that total number of malfunctioning dies after stacking is minimal. Again, we can represent a wafer by a vector, now expressing the state of each die. A vector component will be 1 if the corresponding die is malfunctioning and 0 otherwise. Note that, the number of malfunctioning dies after stacking is equal to the sum of the component-wise maximum of the vectors representing the wafers in that waferstack. Thus, the objective in this problem is identical to our objective function.

In Chapter 4 we introduce Partitioning Vectors into Quadruples. In this problem we are given a set of vectors that need to be partitioned into clusters of four vectors each. A cluster of four vectors (a quadruple) is called a *quad*, and the cost of a quad is the sum of the component-wise maxima of the four vectors in the quad. The goal of the problem is to find a partition with minimum total cost.

We describe a natural, matching-based algorithm, and show that this algorithm is a $\frac{3}{2}$-approximation algorithm for this problem. We also analyze the performance of this algorithm on special cases of the problem. In the most restrictive special case each vector is a $\{0,1\}$-vector containing exactly two ones, and the vectors, when seen as expressing the vertex/edge incidence matrix of a graph, correspond to edges of a simple, connected graph. For this special case, we prove that the problem is NP-hard and that the matching-based algorithm is a $\frac{5}{4}$-approximation algorithm.

2

ROBUST BALANCED OPTIMIZATION

> In theory, there is no difference between theory and practice. But, in practice, there is.
>
> *Jan L.A. van de Snepscheut*

2.1 INTRODUCTION

We investigate balanced optimization problems with vector costs; alternatively, these problems can be described as the robust optimization version of balanced optimization problems. We see this set of problems as a family of optimization problems. We will give the details of this family later, for now we concentrate on a particular problem in this family: the balanced assignment problem with vector costs.

In the balanced assignment problem, see Martello et al. (1984), we are given an $n \times n$ matrix C with real entries $c(i,j)$ for $1 \leqslant i,j \leqslant n$. An *assignment* A is a set of n matrix entries that contains exactly one entry from every row and every column. The *imbalance* of assignment A is given by

$$\max_{(i,j)\in A} c(i,j) - \min_{(i,j)\in A} c(i,j),$$

This chapter is the result of a collaboration with F. C. R. Spieksma and G. J. Woeginger. A journal article based on this chapter has been published in the EURO Journal on Computational Optimization (Ficker et al., 2018). A preliminary version of this chapter has appeared in the Proceedings of the 14th International Workshop on Approximation and Online Algorithms (Ficker et al., 2017).

and the goal is to find an assignment that minimizes the imbalance. In a generalization of this problem, the entries $c(i,j)$ are not real scalars but real vectors $\mathbf{c}(i,j)$ of dimension d; that is

$$\mathbf{c}(i,j) \;=\; (c_1(i,j), c_2(i,j), \ldots, c_d(i,j)), \quad \text{for } 1 \leqslant i,j \leqslant n.$$

The imbalance in the k-th coordinate of assignment A (with $1 \leqslant k \leqslant d$) is

$$\Delta_k(A) \;=\; \max_{(i,j)\in A} c_k(i,j) - \min_{(i,j)\in A} c_k(i,j),$$

and the imbalance of assignment A is finally given by

$$\Delta_{\max}(A) \;=\; \max_k \Delta_k(A).$$

The objective in the *balanced assignment problem with vector costs* is to find an assignment A that minimizes the imbalance $\Delta_{\max}(A)$ (see Figure 1 for an example). Note that for $d = 1$ we recover the traditional balanced assignment problem.

$(3,2,4)$	$(1,2,3)$	$(2,1,1)$
$(3,2,3)$	$(1,4,3)$	$(1,3,3)$
$(2,2,2)$	$(1,2,3)$	$(1,2,4)$

$$A \;= \{(1,2),(2,3),(3,1)\}$$
$$\Delta_{\max}(A) = \max\{(2-1),(3-2),(3-2)\}$$

Figure 1: An instance of the Balanced Assignment Problem with vector costs, with dimension $d = 3$

Balanced optimization problems with vector costs are closely connected to robust optimization (see Section 2.1.1 for a literature review). Robust optimization is a methodology to deal with uncertainty in a problem's coefficients; in the case of discrete scenarios we receive a set of scenarios S, in which each scenario $k \in S$ represents a possible realization of the coefficients. One popular objective in robust optimization is the min-max criterion where we aim to construct a best possible solution in a worst case scenario. Let us argue that, when having a discrete

set of scenarios and using the min-max criterion, robust optimization of balanced optimization problems is, in fact, identical to balanced optimization with vector costs. Consider the robust optimization version of the balanced assignment problem, where cost $c_k(i,j)$ is the cost for assigning i to j in scenario $k \in S$, for $1 \leqslant i,j \leqslant n$. Observe that finding the best balanced assignment for the worst scenario in S (i.e., using the min-max criterion) is equivalent to solving the balanced assignment problem with vector costs (with dimension $d = |S|$). Thus, under the min-max criterion, balanced optimization with vector costs is identical to robust balanced optimization.

Also, there are practical applications of balanced optimization problems with vector costs documented in the literature. As mentioned in Section 1.2.1, Kamura and Nakamori (2014) describe an industrial problem that gives rise to a (specially structured) balanced assignment problem with vector costs; see Section 2.5 for more details on this.

2.1.1 *Related Literature*

On the one hand, there is a stream of literature dealing with robust optimization for traditional (i.e., not balanced) combinatorial optimization problems. On the other hand, there is a stream of literature dealing with balanced optimization problems with scalar costs. Let us first, without attempting to survey the field, consider work in robust optimization for combinatorial optimization problems.

An early contribution is the work by Kouvelis and Yu (1997) who survey the state of the art until 1997. Ben-Tal and Nemirovski (1998) show that for convex optimization problems, the concept of using an ellipsoidal set to model uncertainty, still allows for efficiently solvable optimization problems. This work is extended in Ben-Tal and Nemirovski (1999, 2000), and Ben-Tal et al. (2006). Bertsimas and Sim (2003) show that robust versions of many combinatorial optimization problems (including matching and shortest path) can still be efficiently solved. Here, uncertainty is modeled by specifying an interval for each of the cost coef-

ficients. Another option is to model uncertainty by using settings where a discrete number of scenarios is given (e.g., Aissi et al. (2005)); clearly, the resulting problems are no easier than their deterministic counterparts.

In particular, the robust assignment problem under a fixed number of scenarios is investigated in Deineko and Woeginger (2006); they show that this problem is equivalent to the exact perfect matching problem (whose complexity is an open problem). Poss (2014) shows how dynamic programming can be used to solve a general model for robust combinatorial optimization, where instead of allowing a fixed number of coefficients to deviate from their nominal values, a budget of uncertainty is introduced. Wiesemann et al. (2014) take robust optimization further by developing distributionally robust optimization, a setting where the probability distribution generating data belongs to a so-called ambiguity set. Particular applications of robust optimization are described in Ben-Tal et al. (2005) (logistics), Koster et al. (2013) and Lee et al. (2012) (network design).

Summarizing, there is a wealth of work on robust optimization for combinatorial optimization problems; for an overview of these results we refer to the surveys and books by Aissi et al. (2009), Bertsimas et al. (2011), Ben-Tal et al. (2009), Gabrel et al. (2014) and Gorissen et al. (2015). As far as we are aware, robust versions of balanced combinatorial optimization problems have not been investigated.

Let us now consider work on balanced optimization problems with scalar costs. Martello et al. (1984) introduce a framework containing many balanced optimization problems with scalar costs, and present an algorithm to solve these problems. If the existence of a feasible solution can be decided in polynomial time, the corresponding algorithm is a polynomial time algorithm. We now discuss some of these problems in more detail.

In the balanced version of the shortest path problem, we are given a directed graph $G = (V, E)$, two nodes s and t, and scalar costs on

the edges. The goal is to find a path from s to t that minimizes the difference between the largest and the smallest edge cost along the path. Turner (2012) generalizes this problem to finding a path that minimizes the difference between the k_1-th largest and the k_2-th smallest edge cost, and shows that this problem is solvable in polynomial time. Cappanera and Scutellà (2005) discuss other balanced path problems. Their goal is to identify p (arc-disjoint or node disjoint) paths from s to t, such that the difference between the length of longest path and the length of the shortest path is minimal. These problems are NP-hard, even for $p = 2$.

In the balanced version of the minimum cut problem, we are given an undirected graph $G = (V, E)$, two nodes s and t, and scalar costs on the edges. The goal is to find a cut that minimizes the difference between the largest and the smallest cost of edges in the cut. Katoh and Iwano (1994) construct an algorithm for this problem with running time $O(\text{MST}(|V|, |E|) + |V| \log |V|)$, where $\text{MST}(|V|, |E|)$ denotes the running time for computing the minimum and maximum spanning trees in a graph $G = (V, E)$.

In the balanced version of the spanning tree problem, we are given a graph $G = (V, E)$ and scalar costs on the edges. The goal is to find a spanning tree that minimizes the difference between the largest and the smallest edge cost in the spanning tree. Camerini et al. (1986) and Galil and Schieber (1988) construct algorithms for this problem, with running times $O(|E| \cdot |V|)$ and $O(|E| \log |V|)$ respectively.

In the balanced version of the traveling salesman problem, we are given a graph $G = (V, E)$ and scalar costs on the edges. The goal is to find a Hamiltonion cycle that minimizes the difference between the largest and the smallest edge cost in the cycle. This problem is obviously NP-hard, and Larusic and Punnen (2011) discuss several heuristics for it. Kinable et al. (2017) discuss a related problem, called the equitable traveling salesman problem, where the goal is to find a Hamiltonian cycle in which the difference between the cost of its two matchings is minimal.

Another interesting problem in this area is the balanced version of linear programming. Here we are given a system of linear constraints ($Ax = b$ and $x \geqslant 0$) and costs associated with each real variable x_i. The goal is to minimize the difference between the largest non-zero cost $c_i x_i$ and the smallest non-zero cost $c_j x_j$. Ahuja (1997) presents a polynomial time algorithm for this problem. Balanced optimization problems with an additional linear constraint are treated in Punnen and Nair (1999).

Finally, an example of an optimization problem featuring vector costs is described by Dokka et al. (2014); we stress however that the objective in the underlying multi-index assignment problem is quite different from minimizing imbalance.

2.1.2 *Our Results*

We derive a variety of results on the complexity and approximability of balanced optimization problems with vector costs:

- First, we describe a framework for balanced optimization problems that takes vector costs into account, thereby extending the work of Martello et al. (1984); see Section 2.2.
- Every problem in our framework (i) is solvable in polynomial time if the dimension d is fixed (see Section 2.3.1), and (ii) allows a polynomial time 2-approximation algorithm (see Section 2.3.2).
- For several problems in the framework (among which assignment, spanning tree, s,t-cut, connecting path and Horn-SAT), we prove that the existence of an approximation algorithm with approximation ratio strictly better than 2 implies $P = NP$ (see Section 2.4). Note that these results pinpoint the strongest achievable approximation ratio for these problems (under $P \neq NP$).
- For one problem in our framework (2SAT) we prove that it is actually solvable in polynomial time (see Section 2.4.7). Thus, not all problems in the framework are NP-hard.

- For a special case of the balanced assignment problem with vector costs, namely that problem with vector *sum* costs, we prove that the existence of a polynomial time approximation algorithm with approximation ratio below $4/3$ implies $P = NP$; see Section 2.5.
- We perform extensive computational experiments, investigating the computational behavior of an integer programming formulation and the performance of 2-approximation algorithms on different classes of instances of the robust balanced assignment problem in Section 2.6.

2.2 THE FRAMEWORK

Throughout this chapter, we consider a family of optimization problems that are built around a finite ground set X and a system $\mathcal{F}$ of feasible subsets over X. (The system $\mathcal{F}$ is usually not listed explicitly, but given implicitly in terms of a combinatorial description or in terms of an oracle.) We will only consider problems in this framework, for which the following *feasibility oracle* (as introduced by Martello et al. (1984)) can be performed in time polynomially bounded in the size of X: "Given a subset $Y \subseteq X$, does Y contain a feasible subset from $\mathcal{F}$? And if yes, return a feasible subset of Y from $\mathcal{F}$." Here are some concrete examples of problems that fit this framework:

q-UNIFORM SET SYSTEM. For a given ground set X, a subset $Y \subseteq X$ is feasible if it contains at least q elements of X.

LINEAR ASSIGNMENT. The ground set X are the elements of an $n \times n$ square matrix. A subset $Y \subseteq X$ is feasible if it contains n elements that cover each row and each column of the given matrix.

SPANNING TREE. The ground set X consists of the edges of an undirected graph $G = (V, X)$. A subset $Y \subseteq X$ is feasible if the subgraph (V, Y) contains a spanning tree of G.

s,t-CUT. The ground set X consists of the edges of an undirected graph $G = (V, X)$ with $s, t \in V$. A subset $Y \subseteq X$ is feasible if it contains an

s, t-cut; in other words, the subgraph $(V, E \setminus Y)$ contains no path connecting s and t.

CONNECTING PATH. The ground set X consists of the edges of a directed graph $G = (V, X)$ with $s, t \in V$. A subset $Y \subseteq X$ is feasible if the subgraph (V, Y) contains a path connecting s and t.

2SAT, HORN-SAT. The ground set X consists of all literals both positive and negated of an expression in conjunctive normal form, i.e. $X = \{x_1, \bar{x}_1, \ldots, x_n, \bar{x}_n\}$. A subset $Y \subseteq X$ is feasible if there exists a feasible assignment with the literals in Y. An assignment is feasible if each literal is set to either TRUE or FALSE (either x or $\bar{x}$ is in Y), such that all clauses in the expression are satisfied.

Here is an example of a problem that does NOT fall under this framework (unless P=NP):

HAMILTONICITY. The ground set X consists of the edges of an undirected graph $G = (V, X)$. A subset $Y \subseteq X$ is feasible if the subgraph (V, Y) contains a Hamiltonian cycle.

We will study so-called *robust balanced* versions of the problems in the framework. For this, we generalize the terminology introduced in Section 2.1 in the following way. Besides the ground set X and the system $\mathcal{F}$ of feasible subsets, we introduce a cost function $c : X \to \mathbb{R}^d$ that assigns to every element $x \in X$ a corresponding d-dimensional real vector $\mathbf{c}(x)$; the d coordinates of vector $\mathbf{c}(x)$ will be denoted $c_1(x), \ldots, c_d(x)$. For a subset $Y \subseteq X$, its *imbalance* in the k-th coordinate ($1 \leqslant k \leqslant d$) is defined as:

$$\Delta_k(Y) \;=\; \max_{y \in Y} c_k(y) \;-\; \min_{y \in Y} c_k(y).$$

In other words, this imbalance measures the difference in cost between the largest and smallest value in the k-th coordinate. The *imbalance* of subset Y is finally defined as

$$\Delta_{\max}(Y) \;=\; \max_{1 \leqslant k \leqslant d} \Delta_k(Y). \tag{1}$$

The goal in a robust balanced optimization problem is to find a feasible set Y that minimizes the imbalance $\Delta_{\max}(Y)$. In the sequel, the term "the robust balanced optimization problem" refers to an arbitrary problem in our framework.

2.3 ALGORITHMS FOR ROBUST BALANCED OPTIMIZATION PROBLEMS

In this section we give three algorithms that are applicable to any problem in our general framework. The first algorithm solves the problem in polynomial time when the dimension d of the cost vectors is fixed (Section 2.3.1). The second and third algorithm both yield a 2-approximation in polynomial time (Section 2.3.2). We remind the reader that we only consider problems for which the feasibility oracle can be performed in polynomial time. Throughout this section we use $n := |X|$.

2.3.1 *Fixed Dimension*

We first explain informally the idea behind Algorithm 1; this algorithm generalizes an algorithm presented in Martello et al. (1984).

Recall that a solution to a robust balanced optimization problem consists of some subset of elements of ground set X, each equipped with a cost vector. Now, suppose we would know the smallest value of each coordinate of an optimal solution and its imbalance, without necessarily knowing the elements in the optimal solution. Note that in each coordinate the highest value of the solution is bounded by the sum of the lowest value and the imbalance. This allows us to construct a subset $Y \subseteq X$ consisting of elements whose cost vectors, in every coordinate, lie between the smallest and highest value. Next, applying the feasibility oracle to Y gives us an optimum solution.

Of course, we are not given these values. However, one pair of elements of the ground set allows us to guess the imbalance of an optimal solution. And if the dimension d of the cost vectors is fixed, it is suffi-

cient to try all possibilities for the lowest value of each coordinate in an optimum solution, as is argued in Theorem 2.1.

Algorithm 1

```
1:  Sol := ∞
2:  for each x, y ∈ X let Δ := Δ_max({x, y}) do
3:      for each x_1 ∈ X let min_1 := c_1(x_1) do
4:                  ⋮
5:          for each x_d ∈ X let min_d := c_d(x_d) do
6:              Y := X
7:              for each z ∈ X do
8:                  if ∃k such that c_k(z) < min_k or c_k(z) > min_k + Δ then
9:                      Y := Y \ {z}
10:             if Y contains a feasible solution then
11:                 Sol := min{Sol, Δ}
12: Output Sol.
```

To explain line 2, observe that $\Delta_{\max}(\cdot)$ is defined in (1); further, notice that line 10 is a call to the feasibility oracle.

Theorem 2.1. *Algorithm 1 solves the robust balanced optimization problem in polynomial time, when the dimension* d *of the cost vectors is fixed.*

Proof. Consider an optimal solution with value OPT. This value is determined by two elements from X, and hence, since Algorithm 1 enumerates over all pairs $x, y \in X$, there is a pair x, y with $\Delta_{\max}(\{x, y\}) = \text{OPT}$. Next, by trying out all possibilities for the smallest value in each component k (captured in $\min_k$, see lines 3-5), we are guaranteed to find a solution with value $\Delta_{\max}(\{x, y\})$ if one exists. Hence, Algorithm 1 is exact.

Regarding the complexity of Algorithm 1: for each pair of elements, we consider n^d possibilities for the smallest value in component k ($k = 1, \ldots, d$), and we check for each element in X whether the values of the corresponding cost vector satisfy the resulting bounds. For the resulting set of elements, we call the feasibility oracle to check whether there

exists a feasible solution. Since, by definition, the oracle runs in polynomial time, Algorithm 1 runs in polynomial time (for a fixed d). □

Observe that Algorithm 1 solves the familiar balanced spanning tree, balanced assignment, balanced path, balanced cut problem which all arise when $d = 1$.

2.3.2 *Approximation Algorithms*

When the dimension d of the cost vectors is part of the input, the problem becomes more difficult. Simply trying all possibilities for the lowest value of each coordinate now results in an exponential time algorithm. Instead, we consider every pair of elements of the ground set as a guess for *all* coordinates at the same time. More in particular, we will only consider elements from the ground set that, in every coordinate, do not differ more than $\Delta_{max}(\{x_1, x_2\})$ in some coordinate from either x_1 or x_2. Recall that $\Delta_{max}(\{x_1, x_2\})$ refers to the largest difference over the coordinates between elements $x_1, x_2 \in X$. Doing so gives us a 2-approximation, even when the dimension d of the cost vectors is part of the input.

Algorithm 2

for each pair x_1, x_2 **in** ground set X **do**
  $\Delta := \Delta_{max}(\{x_1, x_2\})$
  $Y := X$
  for each $x \in X$ **do**
    if $\max_k |c_k(x_1) - c_k(x)| > \Delta$ **or** $\max_k |c_k(x_2) - c_k(x)| > \Delta$ **then**
      $Y := Y \setminus \{x\}$
  if Y contains a feasible solution **then**
    $Sol(x_1, x_2) := \Delta_{max}(Y)$
$Sol := \min_{x_1, x_2} Sol(x_1, x_2)$
Output Sol.

Theorem 2.2. *Algorithm 2 is a 2-approximation algorithm for the balanced vector-cost problem.*

Proof. Let OPT denote the imbalance of an optimal solution. By trying out all possible element pairs x_1 and x_2 from the ground set, we will certainly find the two elements in the optimal solution that determine the objective value; in other words, $\Delta_{\max}(\{x_1, x_2\}) = \text{OPT}$.

We remove all elements y from the ground set that satisfy $\Delta_{\max}(\{y, x_1\}) > \Delta$ or $\Delta_{\max}(\{y, x_2\}) > \Delta$; note that these removed elements can never show up in an optimal solution that contains x_1 and x_2 and that has imbalance Δ. Clearly, $\Delta_{\max}(Y)$ is determined by two of its elements, say y_1 and y_2. In other words, there exist $y_1, y_2, \in Y$ such that

$$\Delta_{\max}(Y) = \Delta_{\max}(\{y_1, y_2\}) \leqslant \Delta_{\max}(\{y_1, x_1\}) + \Delta_{\max}(\{y_2, x_1\}) \leqslant 2\Delta.$$

Clearly, this procedure runs in polynomial time: checking whether an element $x \in X$ needs to be removed can be done in $O(d)$ time, and we need to perform the feasibility oracle $O(n^2)$ times. □

Notice that this algorithm also applies to problems for which the feasibility oracle is not solvable in polynomial time. More precisely, let $f(n)$ denote the running time of the feasibility oracle. The running time of Algorithm 2 equals $O\left(n^3 \cdot d + n^2 \cdot f(n)\right)$.

Algorithm 2 compares each element of the ground set with both x_1 and x_2. However, in order to obtain a 2-approximation, it is in fact sufficient to compare only with x_1 (or only with x_2). Using that, we sketch an alternative 2-approximation algorithm, where for each element $x \in X$, we create a set S_x containing at most n possible imbalance values. We then use binary search to find the smallest imbalance for which there still exists a feasible solution, among elements that are not too far from x. It is not difficult to see that the resulting algorithm is also a 2-approximation algorithm. We refer to this algorithm as Algorithm 3.

Notice that this algorithm, compared to Algorithm 2, requires less running time, namely $O\left(n^2 \cdot d + n \log n \cdot f(n)\right)$, where $f(n)$ denotes the running time of the feasibility oracle.

Algorithm 3

Sol := ∞
for each $x \in X$ **do**
  set $S_x := \emptyset$
  for each $y \in X$ **do**
    $S_x = S_x \cup \{\Delta_{max}(\{x, y\})\}$
  Sort S_x in nondecreasing order
  while Binary Search on S_x **do**
    select $\Delta \in S_x$
    $Y := X$
    for each $y \in X$ **do**
      if $\Delta_{max}(\{x, y\}) > \Delta$ **then**
        $Y := Y \setminus \{y\}$
    if Y contains a feasible solution **then**
      if $\Delta <$ Sol **then**
        Sol := Δ
      select $\Delta' \in S_x$ with $\Delta' < \Delta$ using Binary Search rule
    else select $\Delta' \in S_x$ with $\Delta' > \Delta$ using Binary Search rule
Output Sol.

2.4 THE COMPLEXITY OF ROBUST BALANCED OPTIMIZATION

Many balanced optimization problems with scalar costs are known to be solvable in polynomial time (see the discussion in Section 2.1.1): q-Uniform Set Systems, the Linear Assignment problem, the Spanning Tree problem, the s, t-Cut problem, the Connecting Path problem, Horn-SAT and 2SAT. In this section we discuss the complexity of each of these problems in the case when vector costs are given. We show that each of these problems, except robust balanced 2SAT, becomes NP-hard, and that the existence of a polynomial-time $(2-\epsilon)$-approximation algorithm for each of the mentioned problems, except robust balanced 2SAT, implies P=NP. We also show that robust balanced 2SAT is in fact solvable in polynomial time, which shows that not all interesting problems in the framework are NP-hard (Section 2.4.7).

There are three problems, well-known to be NP-complete, that we use in our reductions.

> Problem: INDEPENDENT SET (IS)
> GIVEN: A graph $G = (V, E)$ with vertex set $V = \{v_1, \ldots, v_n\}$ and edge set $E = \{e_1, \ldots, e_m\}$; an integer z.
> QUESTION: Does there exist a subset $I \subseteq V$ with $|I| = z$, such that the vertices in I do not span any edges in G?

> Problem: 3-COLORING
> GIVEN: A graph $G = (V, E)$ with vertex set $V = \{v_1, \ldots, v_n\}$ and edge set $E = \{e_1, \ldots, e_m\}$.
> QUESTION: Does there exist a 3-coloring $f : V \to \{1, 2, 3\}$, such that all edges $(u, v) \in E$ satisfy $f(u) \neq f(v)$?

> Problem: 3SAT
> GIVEN: Set U of variables, collection C of clauses over U such that each clause in C contains at most 3 literals.
> QUESTION: Does there exist a satisfying truth assignment for C?

2.4.1 *Robust Balanced* q*-Uniform Set Systems*

Let us first consider the robust balanced q-Uniform Set System problem. Given a ground set X and an integer q, the robust balanced q-Uniform Set System problem asks for q elements from set X with minimal imbalance.

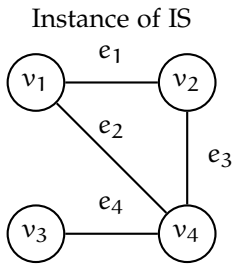
Theorem 2.3. *The robust balanced* q*-Uniform Set System problem is NP-hard.*

Proof. Given an instance of IS represented by a given graph $G = (V, E)$ and an integer z, we construct an instance of robust balanced q-Uniform Set System as follows. The ground set X coincides with the vertex set V of the graph G. A subset $Y \subseteq X$ is feasible if and only if it contains $q := z$ elements. For the definition of the vector costs of X, we turn G into a

directed graph by first choosing some ordering of the vertices in V, and next orienting every edge from the incident vertex with smaller index (source) to the incident vertex with larger index (target). The dimension of the vectors is $d := |E| = m$, and every coordinate k corresponds to a unique edge e_k in E, $1 \leqslant k \leqslant m$. Let us now define cost vector $\mathbf{c}(v_j) = (c_1(v_j), c_2(v_j), \ldots, c_d(v_j))$ corresponding to each vertex $v_j \in V$. For each $v_j \in V$ and $k \in \{1, \ldots, m\}$:

$$c_k(v_j) := \begin{cases} 1 & \text{if vertex } v_j \text{ is the source of the oriented edge } e_k; \\ -1 & \text{if vertex } v_j \text{ is the target of the oriented edge } e_k; \\ 0 & \text{otherwise.} \end{cases}$$

This specifies the corresponding instance of the robust balanced q-uniform set system, see Figure 2 for an illustration.



Corresponding instance of q-Uniform Set System

$\mathbf{c}(v_1) = (1, 1, 0, 0)$
$\mathbf{c}(v_2) = (-1, 0, 1, 0)$
$\mathbf{c}(v_3) = (0, 0, 0, 1)$
$\mathbf{c}(v_4) = (0, -1, -1, -1)$

Figure 2: An instance of robust balanced q-uniform set system

We claim that there exists a feasible subset $Y \subseteq X$ with $|Y| = z$ and $\Delta_{max}(Y) \leqslant 1$ if and only if the considered instance of IS has answer YES.

Assume that there exists a feasible subset $Y \subseteq X$ with $|Y| = z$ and $\Delta_{max}(Y) \leqslant 1$. Suppose for the sake of contradiction that the vertex set corresponding to Y would span some edge $e_k \in E$. Then, in the k-th coordinate, the cost vector of the source vertex of e_k is -1, and the cost vector of the target vertex of e_k is $+1$. Hence $\Delta_{max}(Y) \geqslant 2$. This contradiction shows that Y is a z-element independent set in G.

Next assume that the IS instance has answer YES and let I be the corresponding certificate. Thus $|I| = z$ and in none of the coordinates, the vectors $\mathbf{c}(y)$ with $y \in I$ take the value $+1$ and -1. This yields the desired $\Delta_{\max}(I) \leqslant 1$. □

Corollary 2.1. *The robust balanced* q*-Uniform Set System problem does not allow a polynomial time approximation algorithm with worst case guarantee strictly better than* 2 *(unless P=NP).*

Proof. This is implied by the proof of Theorem 2.3. Indeed, a polynomial time approximation algorithm with a worst case guarantee strictly better than 2, would allow us to distinguish the instances with imbalance at most 1 from the instances with imbalance at least 2. □

2.4.2 *Robust Balanced Linear Assignment*

Given a square matrix C, where each entry is a d-dimensional vector, the robust balanced assignment problem asks for an assignment in C minimizing the imbalance.

Theorem 2.4. *The robust balanced assignment problem is NP-hard.*

Proof. Given an instance of IS represented by a given graph $G = (V, E)$ and an integer z, we construct an instance of the robust balanced assignment problem as follows. Each vertex $v_i \in V$ will correspond to a row $a := i$ in the matrix C ($1 \leqslant i \leqslant n$). The ground set X coincides with the elements of an $n \times n$ matrix C. A subset $Y \subseteq X$ is feasible if it contains n elements that cover each row and each column of matrix C.

For the definition of the cost vectors of X, we turn G into a directed graph by orienting every edge from the incident vertex with smaller index (source) to the incident vertex with larger index (target).

Let us now define the cost vectors $\mathbf{c}(a, b)$, $1 \leqslant a, b \leqslant n$; the dimension of each cost vector equals the number of edges in E, that is $d := m$. We distinguish between entries in the first z columns of C, and entries in the last $n - z$ entries of C as follows:

- For each $b \leqslant z$, we set for each $k = 1, \ldots, m$, $a = 1, \ldots, n$:

$$c_k(a, b) := \left\{ \begin{array}{ll} 1 & \text{if vertex } v_a \text{ is the source of edge } e_k; \\ -1 & \text{if vertex } v_a \text{ is the target of edge } e_k; \\ 0 & \text{otherwise.} \end{array} \right\}$$

- For each $b \geqslant z + 1$, we set: $\mathbf{c}(a, b) := \mathbf{0}$, for each $a = 1, \ldots, n$.

This describes the instance of the robust balanced assignment problem. We now argue equivalence between the existence of an independent set in G of size z, and an assignment in the matrix C with imbalance at most 1.

Suppose the matrix C admits an assignment with imbalance at most 1. Consider the selected entries in the first z columns of C, and consider the corresponding set of rows. Since each row corresponds to a vertex in V, this gives us a set of vertices. We claim that this set of vertices is an independent set, since there can be no coordinate in the cost vectors that corresponds to the selected entries where both a $+1$ and a -1 figures (since the imbalance is at most 1). Thus, there is no arc that goes from one vertex in the set to another vertex in the set, or in other words, we have an independent set of size z.

Now, assume there is an independent set I of size z in the graph G. We propose the following assignment: consider a vertex in I: choose as entry, the corresponding row, and a column with index $b \leqslant z$. The remaining entries are chosen arbitrarily to complete the assignment. Since we have an independent set, the imbalance can be at most 1. □

Corollary 2.2. *The robust balanced assignment problem does not allow a polynomial time approximation algorithm with worst case guarantee strictly better than 2 (unless P=NP).*

2.4.3 *Robust Balanced Spanning Tree*

Given a graph with a cost vector for each edge, the robust balanced spanning tree problem asks for a spanning tree in this graph minimizing the imbalance.

Theorem 2.5. *The robust balanced spanning tree problem is NP-hard.*

Proof. Given an instance of IS represented by a given graph $G = (V, E)$ and an integer z, we construct an instance of the robust balanced spanning tree problem, represented by a graph $H = (V', E')$, as follows. Notice that the ground set X will coincide with the set of edges E'.

For the definition of the cost vectors of X, we turn G into a directed graph by orienting every edge from the incident vertex with smaller index (source) to the incident vertex with larger index (target).

The graph H has a node for each $v_i \in V$, $i \in \{1, \dots, n\}$ plus z additional nodes. The latter set of nodes is referred to as Z, i.e. $V' = V \cup Z$. There is an edge from v_1 to every other $v_i \in V$, and there is an edge from every $v_i \in V$ to each vertex in Z. The latter set of edges shall be referred to as E'_Z. (see also fig. 3).

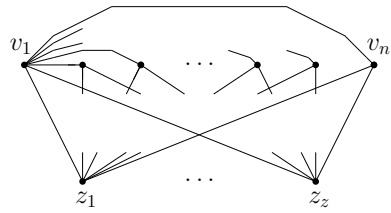


Figure 3: Construction of graph $H = (V', E')$

Let us now specify the cost vector for each edge in E'; the dimension of each cost vector is $|E| + |E'_Z| = m + n \cdot z$. A coordinate corresponds to a unique edge $e_k \in E$ if $k \leqslant m$ or a unique edge $d_k \in E'_Z$ if $k > m$. There are three cases.

- For each $v_1, v_i \in V \setminus \{v_1\}$, we set $\mathbf{c}(v_1, v_i) := \mathbf{0}$.

- For each $v_i \in V$ and $z_j \in Z$ we set, for each $k \leqslant m$:

$$c_k(v_i, z_j) := \begin{cases} 1 & \text{if vertex } v_i \text{ is the source of edge } e_k \in E; \\ -1 & \text{if vertex } v_i \text{ is the target of edge } e_k \in E; \\ 0 & \text{otherwise.} \end{cases}$$

- For each $v_i \in V$ and $z_j \in Z$ we set, for each $k > m$:

$$c_k(v_i, z_j) := \begin{cases} 1 & \text{if edge } d_k = (v_i, z_j); \\ -1 & \text{if edge } d_k = (v_i, z_\ell) \text{ for } z_\ell \in Z \text{ and } j \neq \ell; \\ 0 & \text{otherwise.} \end{cases}$$

We now argue equivalence between existence of an independent set in G of size z and a spanning tree in H with imbalance at most 1.

Suppose that graph H contains a spanning tree with imbalance at most 1. Then every vertex in Z is connected to at least one vertex $v_i \in V'$. It is not possible that one vertex $v_i \in V'$ is connected to two distinct vertices in Z, because there can be no coordinate $k > m$ in the cost vectors that attains both a $+1$ and a -1. This gives us a set of at least z different vertices v_i. In G, no edge exists between any two vertices in this set, since there can be no coordinate in the cost vectors that attains both a $+1$ and a -1. Thus, we have an independent set of size at least z.

Now assume that there is an independent set I of size z in graph G. We propose the following spanning tree: consider a vertex in Z, select the edge connecting it with a vertex $v_i \in I$ (that hasn't been chosen yet). We also select the edges between v_1 and each $v_i \in G$. Since we have an independent set, the imbalance is at most one. □

Corollary 2.3. *The robust balanced spanning tree problem does not allow a polynomial time approximation algorithm with worst case guarantee strictly better than 2 (unless P=NP).*

2.4.4 *Robust Balanced* s, t-*Cut*

Given a graph with a cost vector for each edge, and two nodes s and t, the robust balanced s, t-cut problem asks for a cut in this graph separating s and t that minimizes the imbalance.

Theorem 2.6. *The robust balanced* s, t-*cut problem is NP-hard.*

Proof. Given an instance of IS represented by a given graph $G = (V, E)$ and an integer z, we construct an instance of the robust balanced s, t-cut problem, represented by a graph $H = (V', E')$, as follows. Notice that the ground set X will coincide with the set E'.

The node set V' contains two nodes s, t, and for each $v_i \in V$, there is a layer, layer i, containing z nodes in V', ($i \in \{1, \ldots, n\}$). The z nodes in layer i are denoted by $v_i^1, v_i^2, \ldots, v_i^z$ ($i \in \{1, \ldots, n\}$). There is an edge e_i^j between v_{i-1}^j and v_i^j ($i \in \{2, \ldots, n\}$) for each $j = 1, \ldots, z$. There is an edge e_1^j from s to each node in the first layer, i.e., $e_1^j = \{s, v_1^j\} \in E'$ for each $j = 1, \ldots, z$. And finally, there is an edge e_t^j from each node in the last layer to t, i.e., $e_t^j = \{v_n^j, t\} \in E'$ for each $j = 1, \ldots, z$. Notice that the resulting graph H contains z node-disjoint paths from s to t. (see also fig. 4). Hence, any cut separating s from t will contain at least z edges.

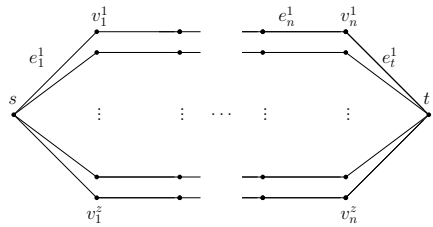


Figure 4: Construction of graph $H = (V', E')$

Let us now specify the cost vector for each edge in E'; the dimension of each cost vector is $|E| + |E'| = m + (n+1) \cdot z$. A coordinate corresponds to a unique edge $e_k \in E$ if $k \leqslant m$ or a unique edge $d_k \in E'$ if $k \geqslant m + 1$.

The vector cost $c(e_i^j)$ of an edge $e_i^j \in E'$ for $i \in \{1, \ldots, n\}, j = 1, \ldots, z$ is defined as follows,

- for each $k \leqslant m$:

$$c_k(e_i^j) := \begin{cases} 1 & \text{if vertex } v_i \text{ is the source of edge } e_k; \\ -1 & \text{if vertex } v_i \text{ is the target of edge } e_k; \\ 0 & \text{otherwise.} \end{cases}$$

- for each $k \geqslant m+1$:

$$c_k(e_i^j) := \begin{cases} 1 & \text{if edge } e_i^j \text{ equals edge } d_k; \\ -1 & \text{if edge } d_k \text{ enters node } v_\ell^j \text{ for } \ell \in \{1,\dots,n\},\ k \neq i; \\ -1 & \text{if edge } d_k \text{ enters node } v_i^\ell \text{ for } \ell \in \{1,\dots,z\},\ \ell \neq j; \\ -1 & \text{if edge } d_k \text{ enters node } t; \\ 0 & \text{otherwise.} \end{cases}$$

The vector cost $\mathbf{c}(e_t^j)$ of edge $e_t^j \in E'$, for $j = 1,\dots,z$ is defined as follows,

- for each $k \leqslant m$ we have $c_k(e_t^j) = 0$.
- for each $k \geqslant m+1$:

$$c_k(e_t^j) := \begin{cases} 1 & \text{if edge } e_t^j \text{ equals edge } d_k; \\ -1 & \text{if edge } d_k \text{ enters node } t,\ d_k \neq e_t^j; \\ 0 & \text{otherwise.} \end{cases}$$

We now argue equivalence between existence of an independent set in G of size z and an s,t-cut in H with imbalance at most 1.

Suppose that graph H contains an s,t-cut C with imbalance at most 1. Since we have z parallel paths in H, we know that $|C| \geqslant z$. The cost vectors of the edges in H are constructed in such a way that (i) choosing two edges from an s,t-path in H in the cut, would lead to imbalance at least 2, (ii) choosing two edges from the same layer (edges e_i^j and e_ℓ^j, with $j \neq \ell$) in H would lead to imbalance at least 2 and (iii) choosing

two edges of which one contains node t, would lead to imbalance at least 2. It follows that C cannot contain two edges from the same path, neither can it contain two edges of the same layer, nor can it contain an edge entering node t. Hence, cut C consists of exactly z edges e_i^j, each associated with a distinct $v_i \in V$. It is a fact that the corresponding set of vertices in G is an independent set in G, since there can be no edge connecting any of these z vertices, because there is no coordinate $k \leqslant m$ in the cost vectors that attains both a $+1$ and a -1.

Now assume that there is an independent set I of size z in graph G. Without loss of generality we may rename the nodes such that $I = \{v_1, \ldots, v_z\}$. We select edges e_j^j ($j \in \{1, \ldots, z\}$) to be in the cut. This is indeed an s,t-cut, because we take exactly one edge from each parallel path. Since these z edges are associated with distinct nodes (in V) and correspond to a single edge from every path in H, the imbalance is at most one. □

Corollary 2.4. *The robust balanced s,t-cut problem does not allow a polynomial time approximation algorithm with worst case guarantee strictly better than 2 (unless P=NP).*

2.4.5 *Robust Balanced Connecting Path*

Given a graph with a cost vector for each edge, and two nodes s and t, the robust balanced connecting path problem asks for a path connecting s and t that minimizes the imbalance.

Theorem 2.7. *The robust balanced connecting path problem is NP-hard.*

Proof. Given an instance of 3-COLORING represented by a graph $G = (V, E)$, we construct an instance of the robust balanced connecting path problem, represented by a graph $H = (V', A)$, as follows. Notice that the ground set X will coincide with the set A.

First, we turn G into a directed graph by first choosing some ordering of the vertices in V, and next orienting every edge from the incident

vertex with smaller index (source) to the incident vertex with larger index (target). Next, the graph H has a node s, a node t, and for each $v_i \in V$ ($i \in \{1, \ldots, n\}$), there is a layer of three nodes, called the top node, the middle node, and the bottom node in H. There is an arc from each node in layer i to each node in layer $i+1$, ($1 \leqslant i \leqslant n-1$). Node s is connected to each node of layer 1, and each node of layer n is connected to node t. There are no other arcs (see also fig. 5).

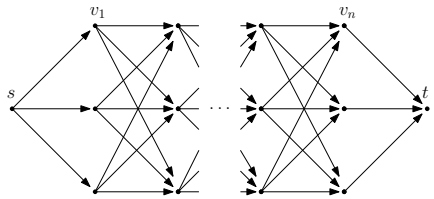


Figure 5: Construction of graph $H = (V', A)$

Let us now specify the cost vector that corresponds to each arc $a \in A$; there are three coordinates corresponding to each edge $e_k \in E$, hence, its dimension equals $3m$. The vector-cost $\mathbf{c}(a)$ of an arc $a \in A$ is defined as follows:

- For an arc a entering a *top node* in layer i ($1 \leqslant i \leqslant |V|$) for $k = 1, \ldots, m$

$$c_{3k-2}(a) := \begin{cases} 1 & \text{if edge } v_i \text{ is the target of oriented edge } e_k; \\ -1 & \text{if edge } v_i \text{ is the source of oriented edge } e_k; \\ 0 & \text{otherwise.} \end{cases}$$

$$c_{3k-1}(a) = 0.$$
$$c_{3k}(a) = 0.$$

- For an arc a entering a *middle node* in layer i ($1 \leqslant i \leqslant |V|$) for $k = 1, \ldots, m$

$$c_{3k-1}(a) := \begin{cases} 1 & \text{if edge } v_i \text{ is the target of oriented edge } e_k; \\ -1 & \text{if edge } v_i \text{ is the source of oriented edge } e_k; \\ 0 & \text{otherwise.} \end{cases}$$

$$c_{3k-2}(a) = 0.$$
$$c_{3k}(a) = 0.$$

- For an arc a entering a *bottom node* in layer i ($1 \leqslant i \leqslant |V|$) for $k = 1, \ldots, m$

$$c_{3k}(a) := \begin{cases} 1 & \text{if edge } v_i \text{ is the target of oriented edge } e_k; \\ -1 & \text{if edge } v_i \text{ is the source of oriented edge } e_k; \\ 0 & \text{otherwise.} \end{cases}$$

$$c_{3k-2}(a) = 0.$$
$$c_{3k-1}(a) = 0.$$

- For an arc a entering node t,

$$\mathbf{c}(a) = 0.$$

We claim that there exists a path between s and t in the graph H with imbalance at most 1 if and only if the graph G is 3-colorable.

Assume that there exists a path between s and t in the graph H with imbalance at most 1. Note that the path contains exactly one node from each level i, $1 \leqslant i \leqslant n$. We color the nodes as follows: if the path contains the top (middle, bottom) node of level i we color node v_i with color 1 (2, 3) ($1 \leqslant i \leqslant |V|$). The imbalance of this path is at most 1, hence no coordinate contains a $+1$ and a -1. Thus no two nodes adjacent in H receive the same color. Hence, the graph is 3-colorable.

Next assume that the graph G is 3-colorable. Then the color of each node in G determines which node in each layer is in the path from s to t: specifically if node v_i is colored with color 1 (2, 3) then the top (middle, bottom) node of layer i is in the path from s to t ($1 \leqslant i \leqslant |V|$). Observe that specifying the nodes that are in the path P gives us the corresponding cost vectors, and moreover, since the coloring of G is such that no edge connects two vertices with the same color, there is no coordinate where there is a $+1$ and a -1. This yields the desired $\Delta_{\max}(P) \leqslant 1$. □

Corollary 2.5. *The robust balanced connecting path problem does not allow a polynomial time approximation algorithm with worst case guarantee strictly better than 2 (unless P=NP).*

2.4.6 *Robust Balanced Horn-SAT*

Given a set of literals X with vector-costs and a set of clauses C', where each clause contains at most 1 positive literal, the robust balanced Horn-SAT problem asks to find a satisfying truth assignment which minimizes the imbalance.

Theorem 2.8. *The robust balanced Horn-SAT problem is NP-hard.*

Proof. Given an instance of 3SAT represented by a set of variables $U = \{u_1, \ldots, u_n\}$ and a set of clauses $C = \{c_1, \ldots, c_m\}$, we construct an instance of the robust balanced Horn-SAT problem as follows. The ground set X contains 4 literals for each variable $u_i \in U$, called x_i, $\bar{x}_i$, y_i and $\bar{y}_i$. For each $c_j \in C$ we create a clause $c'_j \in C'$ as follows. If $\bar{u}_i \in c_j$ ($1 \leqslant i \leqslant n$, $1 \leqslant j \leqslant m$) then we have $\bar{x}_i \in c'_j$, and if $u_i \in c_j$ then we have $\bar{y}_i \in c'_j$. For example a clause $(u_1 \vee \bar{u}_2 \vee u_3)$ becomes clause $(\bar{y}_1 \vee \bar{x}_2 \vee \bar{y}_3)$. Note that all clauses $c' \in C'$ only contain negative literals, hence we have an instance of Horn-SAT.

Let us now define the cost vectors for every element in the ground set. The dimension of the vector costs for X is $d = 2n$ and for every k ($1 \leqslant k \leqslant n$) coordinate k corresponds to a literal u_i and coordinate $n + k$ corresponds to literal $\bar{u}_i$. We distinguish 4 cases, one case for each literal corresponding to variable u_i ($1 \leqslant i \leqslant n$).

- For literal x_i, we set for each $k = 1, \ldots, 2n$:

$$c_k(x_i) := \begin{cases} 1 & \text{if literal } x_i \text{ corresponds to literal } u_i \text{ i.e. } k = i; \\ 0 & \text{otherwise.} \end{cases}$$

- For literal y_i, we set for each $k = 1, \ldots, 2n$:

$$c_k(y_i) := \begin{cases} -1 & \text{if literal } y_i \text{ corresponds to literal } u_i \text{ i.e. } k = i; \\ 0 & \text{otherwise.} \end{cases}$$

- For literal $\bar{x}_i$, we set for each $k = 1, \ldots, 2n$:

$$c_k(\bar{x}_i) := \begin{cases} 1 & \text{if literal } \bar{x}_i \text{ corresponds to literal } \bar{u}_i \text{ i.e. } k = n + i; \\ 0 & \text{otherwise.} \end{cases}$$

- For literal $\bar{y}_i$, we set for each $k = 1, \ldots, 2n$:

$$c_k(\bar{y}_i) := \begin{cases} -1 & \text{if literal } \bar{y}_i \text{ corresponds to literal } \bar{u}_i \text{ i.e. } k = n + i; \\ 0 & \text{otherwise.} \end{cases}$$

This describes the instance of robust balanced Horn-SAT. We now argue equivalence between the existence of satisfying truth assignment of 3SAT with a satisfying truth assignment for Horn-SAT with imbalance at most 1.

Suppose we have a satisfying truth assignment for 3SAT. If $u_i \in U$ is set TRUE, then we set x_i TRUE and y_i FALSE. If $u_i \in U$ is set FALSE, then we set x_i FALSE and y_i TRUE. Clearly the imbalance of this assignment is 1, as x_i and y_i aren't simultaneously TRUE or simultaneously FALSE. We now argue that this indeed is a satisfying truth assignment for the Horn-SAT instance. Let us consider a clause $c_j \in C$ and let variable u_i be the variable whose assignment satisfies clause c_j. If u_i occurs positive and u_i is TRUE, i.e. $u_i \in c_j$, then we know that $\bar{y}_i \in c'_j$ and y_j is FALSE. That implies that clause $c'_j \in C'$ is also satisfied. If u_i occurs negative and u_i is FALSE, i.e. $\bar{u}_i \in c_j$, then we know that $\bar{x}_i \in c'_j$ and x_i is FALSE. That too implies that clause $c'_j \in C'$ is also satisfied.

Suppose we have a satisfying truth assignment for Horn-SAT with imbalance at most 1. Then we know that x_i and y_i aren't simultaneously TRUE or simultaneously FALSE. If x_i is TRUE and y_i is FALSE then we set $u_i \in U$ TRUE. If x_i is FALSE and y_i is TRUE then we set $u_i \in U$ FALSE. We now argue that this indeed is a satisfying truth assignment for the 3SAT instance. Let us consider a clause $c'_j \in C'$, note that it only contains negated variables, x or y. Let variable x_i be the variable whose assignment satisfies clause c'_j, i.e $\bar{x}_i \in c'_j$ and x_i is FALSE. Then we know that $\bar{u}_i \in c_j$ and u_i is FALSE. That implies that clause $c_j \in C$

is satisfied. Let variable y_i be the variable whose assignment satisfies clause c'_j, i.e $\bar{y}_i \in c'_j$ and y_i is FALSE. Then we know that $u_i \in c_j$ and u_i is TRUE. That too implies that clause $c_j \in C$ is satisfied. □

Corollary 2.6. *The robust balanced Horn-SAT problem does not allow a polynomial time approximation algorithm with worst case guarantee strictly better than 2 (unless P=NP).*

2.4.7 *Robust Balanced 2SAT*

Recall that an instance of the robust balanced 2SAT problem consists of

i) an expression C in conjunctive normal form, where each clause consists of at most two literals, and

ii) a cost vector for each positive literal and each negative literal.

The objective is to find a satisfying truth assignment with minimal imbalance. We show that this problem, unlike the previous problems, is easy.

Theorem 2.9. *The robust balanced 2SAT problem is solvable in polynomial time.*

Proof. First, we prove that we can decide in polynomial time whether a solution with imbalance Δ exists. Consider the cost vectors of each pair of elements $x_1, x_2 \in X$. If there is a coordinate in which these two vectors differ more than Δ, then these two elements cannot occur together in a solution with imbalance Δ. Thus, for each such pair $x_1, x_2 \in X$ we add to the current expression C the clause $(\bar{x}_1 \vee \bar{x}_2)$. (Notice that the negation of a negated literal results in a positive literal, i.e. $\overline{\bar{x}} = x$.)

Observe that this procedure ensures that the resulting instance is also a 2SAT instance, with a size polynomial in the input, and that each feasible solution to this new instance is a feasible solution to the original problem with imbalance at most Δ.

We know that the imbalance Δ of any feasible solution is defined by two elements of the ground set X. That gives us at most $O(n^2)$ distinct

possible values for Δ (one for each pair of elements). The lowest value of Δ for which there exists a truth assignment is the value of an optimal solution. □

2.5 A SPECIAL CASE: ROBUST BALANCED ASSIGNMENT WITH SUM COSTS

Kamura and Nakamori (2014) consider a highly structured special case of the robust balanced assignment problem: the cost vector for every matrix entry $\mathbf{c}(a, b)$ is the sum of two d-dimensional cost vectors $\mathbf{c}(a)$ and $\mathbf{c}(b)$. We call this setting the robust balanced assignment problem with *sum* costs. The resulting problem remains NP-hard, as witnessed by the following result.

Theorem 2.10. *The robust balanced assignment problem with sum costs is NP-hard.*

Proof. We modify the hardness construction used in the proof of Theorem 2.4 as follows. Given an instance of IS represented by a given graph $G = (V, E)$ and integer z, we construct an instance of the robust balanced assignment problem with sum costs as follows. There is an $(n + z) \times (n + z)$ matrix C, the entries of which are the elements of the ground set X. Each vertex $v_i \in V$ will correspond to a row $a := i$ and column $b := i$ in the matrix C $(1 \leqslant i \leqslant n)$. The other z rows $a := n + j$ and z columns $b := n + j$ in the matrix C $(1 \leqslant j \leqslant z)$ are referred to as dummy rows, columns respectively.

We need to specify a d-dimensional cost vector $\mathbf{c}(a)$, for each row a of C, as well as a d-dimensional cost vector $\mathbf{c}(b)$ for each column b of C. The dimension of each cost vector equals the number of edges in G plus 1, i.e., $d := m + 1$.

For the definition of the vector costs, we turn G into a directed graph by orienting every edge from the incident vertex with smaller index (source) to the incident vertex with larger index (target). Let us first

define the cost vectors $\mathbf{c}(a)$ for each row a, $1 \leqslant a \leqslant n+z$. There are two cases:

- For each $a \leqslant n$, we set for each $k = 1, \ldots, m+1$:

$$c_k(a) := \begin{cases} 1 & \text{if vertex } v_a \text{ is the source of edge } e_k; \\ -1 & \text{if vertex } v_a \text{ is the target of edge } e_k; \\ 0 & \text{otherwise.} \end{cases}$$

- For $n+1 \leqslant a \leqslant n+z$, we set for each $k = 1, \ldots, m+1$:

$$c_k(a) := \begin{cases} 0 & \text{if } k \leqslant m; \\ 1 & \text{otherwise.} \end{cases}$$

Let us now define the cost vectors $\mathbf{c}(b)$ for each column b, $1 \leqslant b \leqslant n+z$. There are two cases:

- For each $b \leqslant n$, we set for each $k = 1, \ldots, m+1$:

$$c_k(b) := \begin{cases} -1/2 & \text{if vertex } v_b \text{ is the source of edge } e_k; \\ 1/2 & \text{if vertex } v_b \text{ is the target of edge } e_k; \\ -1 & \text{if } k = m+1; \\ 0 & \text{otherwise.} \end{cases}$$

- For each $n+1 \leqslant b \leqslant n+z$, we set $\mathbf{c}(b) := \mathbf{0}$.

The resulting matrix C can be represented as follows;

$$C = \begin{array}{c} \\ 1 \\ \vdots \\ n \\ n+1 \\ \vdots \\ n+z \end{array} \begin{array}{c} \begin{array}{ccc|ccc} 1 & \ldots & n & n+1 & \ldots & n+z \end{array} \\ \left(\begin{array}{ccc|ccc} & & & & & \\ & & & & & \\ & C_1 & & & C_2 & \\ & & & & & \\ \hline & & & & & \\ & C_3 & & & C_4 & \\ & & & & & \end{array}\right) \end{array}.$$

We now show that the existence of an IS with size z implies the existence of a solution with imbalance at most $3/2$, whereas the non-existence of a size z independent set leads to assignments with imbalance at least 2.

Let us first assume that there is an independent set with size z. We construct the following assignment. Consider a vertex in V. There are two possibilities.

If this vertex is in the independent set, choose as entry in C, the corresponding row a ($a \leqslant n$), and any column $b \geqslant n+1$ (notice that the resulting entry is in block C_2). Using the definition of the cost vectors given in the construction, we find that:

$$c_k(a,b) := \begin{cases} 1 & \text{if vertex } v_a \text{ is the source of edge } e_k; \\ -1 & \text{if vertex } v_a \text{ is the target of edge } e_k; \\ 0 & \text{otherwise.} \end{cases}$$

If this vertex is not in the independent set, then choose as entry the corresponding row a ($a \leqslant n$), and the same column, i.e., $b = a$ (notice that the resulting entry is in block C_1). We find that:

$$c_k(a,a) := \begin{cases} 1/2 & \text{if vertex } v_a \text{ is the source of edge } e_k; \\ -1/2 & \text{if vertex } v_a \text{ is the target of edge } e_k; \\ -1 & k = m+1 \\ 0 & \text{otherwise.} \end{cases}$$

We also choose the following entries for each of the dummy rows: the z columns (each representing a vertex) that have not yet been used (the resulting entries are in block C_3). Then, we have:

$$c_k(a,b) := \begin{cases} -1/2 & \text{if vertex } v_b \text{ is the source of edge } e_k; \\ 1/2 & \text{if vertex } v_b \text{ is the target of edge } e_k; \\ 0 & \text{otherwise.} \end{cases}$$

This specifies the assignment. Observe that for these selected entries it is true that the corresponding cost vectors have, in no coordinate, a $+1$ and a -1; hence, the imbalance is at most $3/2$.

Suppose no size z independent set exists. We claim that any assignment has imbalance $\geqslant 2$. First suppose that the assignment uses an entry in block C_4 of matrix C. Since (w.l.o.g.) $z < n$ any assignment also uses entries in block C_1. But then the last coordinate of the selected vectors leads to an imbalance $\geqslant 2$. Now, suppose no entries from block C_4 are chosen. Then, since one entry in each of the last z columns must be chosen, z entries are chosen from block C_2. Consider the z vertices that correspond to the rows of these entries. Since the set of vertices is not an independent set, there exists an edge e_k between two of these vertices. Hence, the imbalance $\geqslant 2$. □

Notice that this construction does not completely close the gap between the factor of 2 achieved by the approximation algorithms, and what might be achieved by any polynomial time algorithm. We can only state:

Corollary 2.7. *The robust balanced assignment problem with sum costs does not allow a polynomial time approximation algorithm with worst case guarantee strictly better than $4/3$ (unless P=NP).*

Remark 1: Corollary 2.7 leaves open the possibility that Algorithm 2 and/or Algorithm 3 have a better approximation ratio for the robust balanced assignment problem with sum costs. For Algorithm 3 however, this is not the case, as is shown by the following instance.

Suppose we have $n = 3$, and we have row costs $a_1 = (-1,-2)$, $a_2 = (-1,-1)$ and $a_3 = (0,-1)$ and column costs $b_1 = (1,1)$, $b_2 = (2,2)$, and $b_3 = (0,0)$. The resulting cost matrix C is as follows.

(0,-1)	(1,0)	(-1,-2)
(0,0)	(1,1)	(-1,-1)
(1,0)	(2,1)	(0,-1)

Since it is not difficult to see that a solution with value 0 does not exist, it follows that the solution consisting of the elements $\{(1,2),(2,1),(3,3)\}$ with respective cost vectors $\{(1,0),(0,0),(0,-1)\}$, which has value (or imbalance) 1, is an optimal solution.

Let us now run Algorithm 3 on this instance. We show that for any $x \in X$, with an imbalance of 1, the algorithm can return a feasible solution with value equal to 2. First, notice that for $\mathbf{c}(x) \in \{(-1,-2),(-1,-1),(1,1),(2,1)\}$ no feasible solution exists when the elements that differ more than 1 from $\mathbf{c}(x)$ in a component, are removed. Next, for $\mathbf{c}(x) = (0,-1)$ the algorithm can select $\mathbf{c}(1,2) = (1,0)$, $\mathbf{c}(2,3) = (-1,-1)$ and $\mathbf{c}(3,1) = (1,0)$, resulting in a solution with value 2. Finally, for $\mathbf{c}(x) \in \{(0,0),(1,0)\}$ the algorithm can select $\mathbf{c}(1,1) = (0,-1)$, $\mathbf{c}(2,2) = (1,1)$ and $\mathbf{c}(3,3) = (0,-1)$, again leading to a solution with value 2.

Hence, Algorithm 3 can return a solution with imbalance 2, which is twice the optimum.

Remark 2: Given the application described in Kamura and Nakamori (2014), one could be interested in the robust balanced 3-dimensional assignment problem. In this problem, we are given three sets of vectors, say a set A, B and C. Then, the ground set X consists of triples, each consisting of a vector from A, a vector from B, and a vector from C, and the cost vector corresponding to an element from X is nothing else but the sum of the three vectors. Although this problem does not fall in our framework (the feasibility question is NP-hard), one might wonder about the approximability of this robust balanced 3-dimensional assignment problem. We point out, however, that no constant-factor approximation algorithm can exist (unless P=NP), even when $d = 1$.

Theorem 2.11. *The robust balanced 3-dimensional assignment problem does not allow a polynomial time constant-factor approximation algorithm (unless P=NP), even when* $d = 1$.

Proof. There is a straightforward reduction from Numerical 3-Dimensional Matching. Recall that in Numerical 3-Dimensional Matching we are given disjoint sets W, Y and Z, each containing n elements and a cost

$c(a)$ for each $a \in W \cup Y \cup Z$ and a bound b. The goal is to select n pairwise disjoint triples from $W \times Y \times Z$, referred to as M, such that for each of the selected triples, (w, y, z), it holds that $c(w) + c(y) + c(z) = b$.

By having an element in A (B, C), for each element in W (Y, Z) an equivalent instance of the robust balanced 3-dimensional assignment problem arises (note that ground set $X = A \times B \times C$). Notice that the imbalance in this instance is 0 if and only if there exists a set of triples M such that for each triple (w, y, z) in M it holds that $c(w) + c(y) + c(z) = b$. Distinguishing in polynomial time whether the imbalance is 0 or not would imply $P = NP$. □

2.6 COMPUTATIONAL EXPERIMENTS FOR THE ROBUST BALANCED ASSIGNMENT PROBLEM

In this section we describe a computational experiment for the robust balanced assignment problem. Recall that in this problem, we are given a square matrix C, in which each entry is a d-dimensional vector, and we are interested in finding an optimum robust balanced assignment, i.e., in finding an assignment in C minimizing the imbalance (see Section 2.1).

Our goal is to investigate the performances of Algorithm 2 and Algorithm 3 (Section 2.3.2). We also compare the performance of both algorithms to the performance of a Mixed Integer Program (MIP) used to find an optimal solution.

2.6.1 *MIP Formulation*

The following MIP formulation is given by Kamura and Nakamori (2014), and uses parameters $c_k(i, j)$, each of which refers to the k-th coordinate of vector $\mathbf{c}(i, j)$ in matrix C. Further, binary variables $x(i, j)$ indicate whether or not the solution contains entry (i, j), $1 \leqslant i, j \leqslant n$, real variables u_k and ℓ_k denote the respectively largest and smallest value in

dimension k of a selected entry ($1 \leqslant k \leqslant d$), and the real variable t captures the objective function value.

$$\begin{aligned}
&\text{minimize} && t && && (2a)\\
&\text{subject to} && \sum_{i=1}^{n} x_{i,j} = 1 && j = 1,\dots,n, && (2b)\\
& && \sum_{j=1}^{n} x_{i,j} = 1 && i = 1,\dots,n, && (2c)\\
& && c_k(i,j)\cdot x_{i,j} \leqslant u_k && i,j = 1,\dots,n; \forall k, && (2d)\\
& && \ell_k \leqslant c_k(i,j)\cdot x_{i,j} + M(1-x_{i,j}) && i,j = 1,\dots,n; \forall k && (2e)\\
& && u_k - \ell_k \leqslant t && k = 1,\dots,d, && (2f)\\
& && x_{i,j} \in \{0,1\} \quad i,j = 1,\dots,n, && && (2g)\\
& && u_k, \ell_k, t \geqslant 0 \quad k = 1,\dots,d. && && (2h)
\end{aligned}$$

Constraints (2b) and (2c) ensure that an assignment in matrix C is found. Further, constraints (2d) and (2e) imply that the variables u_k and ℓ_k receive their intended values, and constraint (2f) together with the objective function (2a) ensure that t equals the minimum imbalance.

2.6.2 *Construction of Data-sets*

We have created three classes of instances of the robust balanced assignment problem. In Class 1, we construct instances by drawing each individual cost coefficient from a particular probability distribution; instances of Classes 2 and 3 feature scenarios, where the k-th component of each cost vector corresponds to scenario k, $k = 1,\dots,d$. Let us now describe the three classes in more detail.

Class 1

To generate instances of Class 1, we vary the following three parameters:

- n, the number of rows (or columns) of matrix C. We use $n \in \{10, 20\}$.
- d, the length of each cost vector. We use $d \in \{2, 100, 300\}$.
- g, the probability distribution from which the cost coefficients are generated. We use $g \in \{U(0, 100), U(0, 1000), N(50; 1), N(50; 10), N(500; 1), N(500; 10)\}$.

Here $U(a, b)$ stands for a discrete uniform distribution between a and b, while $N(a; b)$ stands for a normal distribution with mean a and standard deviation b. This gives $2 \times 3 \times 6 = 36$ types of instances of Class 1. For each of these types, we have generated 10 instances, leading to a total of 360 instances of Class 1. Solving instances of Class 1 allows us to assess how difficult in practice these instances are, depending upon the various parameters. We study the performance of the MIP-formulation, and Algorithms 2 and 3, both in terms of solution quality and running times.

Class 2

Instances of Class 2 are based on the instances of Class 1 as follows. For each instance of Class 1 we construct an instance of Class 2 by rearranging the cost coefficients of each cost vector in the following way. Given a cost vector, we sort the cost coefficients by their distance to the mean (of the probability distribution we generate from) in an increasing way. Thus, after this rearrangement it holds that the larger the difference with the mean, the greater the index of the resulting component. The idea behind this construction is to investigate a set of scenarios, where the first scenario can be seen as the most "average" scenario, and the last scenario is the most "extreme" scenario; in between, the scenarios become more variable with their index. Clearly, we have 360 instances of Class 2.

Solving instances of Class 2 allows us to see whether this concept of scenarios ranging from average to extreme has an impact on the relative solution times of our methods. It is also interesting to see how the

optima, as well as the computation times, compare to those found for Class 1.

Class 3

As in Class 2 instances, we use the idea of scenarios to generate instances of Class 3. Now however, we generate the cost coefficients of a cost vector from distinct distributions whose variance increases with the component of the vector. More concrete, we vary the following parameters

- n, the number of rows (or columns) of matrix C. We use $n \in \{10, 20\}$.
- d, the length of each cost vector. We use $d \in \{2, 100, 300\}$.
- $g_{k,d}$, the probability distribution from which cost coefficient k, for vector-length d, is generated from ($k = 1, \ldots, d$). For each $d \in \{2, 100, 300\}$, we use the following four collections of probability distributions.
  - $U(\lceil 50 - \frac{50k}{d} \rceil, \lfloor 50 + \frac{50k}{d} \rfloor)$,
  - $U(\lceil 500 - \frac{500k}{d} \rceil, \lfloor 500 + \frac{500k}{d} \rfloor)$,
  - $N(50; \frac{50k}{3d})$, and
  - $N(500; \frac{500k}{3d})$.

  Observe that, for each of these collections, the standard deviation increases with the component index k. For instance, in case of the second collection of probability distributions, with $d = 300$, the first component is drawn from $U(499, 501)$, while the last component is drawn from $U(0, 1000)$, and the components in between come from a uniform distribution whose standard deviation stepwise increases.

Solving instances of Class 3 allows us to see whether the idea of different scenarios coming from different distributions leads to other instances in terms of solvability.

We have used R 3.1.1 to generate the data.

2.6.3 *Details of Implementation*

The MIP and both approximation algorithms are implemented using (free) programming language Julia 0.5.1, together with notebook environment Jupyter (IJulia). We also use package JuMP for Mathematical Optimization (for more information see Dunning et al. (2017)) together with CPLEX 12.7.0 as the solver. Implementing both the MIP and the approximation algorithms in Julia enables us to use a consistent concept of running time to compare. Experiments are run on a laptop with Intel Core i7-4800MQ CPU @2.70GHz and 16GB RAM.

All instances and the Julia Notebooks are available online at https://github.com/AFicker/RobustBalancedAssignment.

For implementing the MIP formulation we have to choose an M that is sufficiently large (see constraints (2e)). If the data from an instance is from a Uniform distribution $U(0, x)$, we set $M = x + 1$, which is greater than the largest possible value. If the data from an instance is from a Normal distribution, we do not know what the largest possible value can be. Hence we look for the highest number h occurring in the dataset, i.e. $h = \max_{i,j,k} c_k(i, j)$, and set $M = h + 1$.

For both approximation algorithms we have to implement the feasibility oracle. For the robust balanced assignment problem this means solving a max-weight assignment restricted to the entries of C that have remained in the set Y (see Section 2.3.2). We do this by creating a $n \times n$ matrix O, setting the entry $o(i, j)$ in O equal to 1 if element $(i, j) \in Y$, and 0 if otherwise. We then solve the max-weight assignment problem as an IP using a call to CPLEX from Julia to solve this instance. If the optimal solution has value n, then Y contains a feasible assignment, otherwise Y does not. Moreover, in case Y allows a feasible assignment, instead of setting $\mathrm{Sol}(x_1, x_2) := \Delta_{\max}(Y)$, we set $\mathrm{Sol}(x_1, x_2)$ equal to the value of the assignment found by the feasibility oracle (since we have an actual solution at our disposal).

Recall that both approximation algorithms are valid for any robust balanced optimization problem; we now describe a modification that we employ in order to tailor both algorithms for the robust balanced assignment problem.

Consider Line 1 in Algorithm 2: selecting a pair of elements to compute a bound/guess on the solution value, and consider Line 3 in Algorithm 3: selecting a second element to compute a bound on the solution value. Since we only need to check those pairs of entries that can actually occur together in a feasible assignment, we do not select pairs of elements from the ground set that either occur in the same row or in the same column.

2.6.4 *Results*

Hereunder we present five tables that describe our computational results. Tables 1- 3 contain the results of instances belonging to class 1, Tables 4- 6 show the results of instances belonging to class 2, and Tables 7 and 8 show the results of instances belonging to class 3. Each table consists of several multicolumns. One multi-column corresponds to the instances generated from one probability distribution, and contains the results for the MIP, Algorithms 2 and 3. Each entry in a table is the average over 10 instances. We present:

- the average computing time in seconds (time $\bar{x}$) and its standard deviation (s),
- the average solution value that was found (Sol $\bar{x}$),
- the average coordinate in which this solution value is attained ($\arg_k \bar{x}$)
- and for Algorithms 2 and 3 we show the average gap compared to the solution found by the MIP (gap(%) $\bar{x}$), its standard deviation (s) and the number of instances for which the gap is 0 (#0).

Note that we interrupted Cplex after one hour of computing time; in that case we report the number of instances (out of 10) that were solved within 3600 seconds and the data of the best found solution. Hence it is possible that there are instances for which the MIP did not find the best possible solution. There is exactly one instance out of 960 for which Algorithm 2 found a better solution than MIP (Algorithm 3 never did), namely for Class 1, $U(0, 1000)$, $n = 20$, $d = 300$ the sixth instance.

Let us first consider the MIP. All instances with $n = 10$ are solved fast; there is, however, a clear dependence on d. Instances with $n = 10$ and $d = 300$ take on average 60 seconds to solve. For $n = 20$, instances with $d = 100$ are often not solved to optimality by the MIP, and, except for instances of Class 2, instances with $n = 20$ and $d = 300$ cannot be solved within 3600 seconds.

Of course, the approximation algorithms fare better than the MIP in terms of computation times: each of them always finds a solution within 3600 seconds. It is also clear that (except for the smallest instances), Algorithm 3 is an order of magnitude faster than Algorithm 2. This speedup becomes more pronounced as the instance becomes larger. For the largest instances, Algorithm 3 still finds a feasible solution within, approximately, 80 seconds, while Algorithm 2 may need more than 2000 seconds.

For future research it might be interesting to try and speed up the MIP. For example one could use binary search together with the MIP-solver to find the smallest imbalance for which there exists a feasible solution. The objective of the MIP is then to find a solution with a given imbalance.

When we turn to the quality of the solutions found by the approximation algorithms, it is clear that these solutions are much better than the worst-case bound may suggest.

Algorithm 2 produces, on average, better solutions than Algorithm 3. For the instances generated by a uniform distribution, Algorithm 2 (Algorithm 3) finds solutions within 7% (12%) of the optimum value, while

Algorithm 2 (Algorithm 3) finds solutions within 14% (21%) of the optimum value for instances generated by the normal distribution. Furthermore, Algorithm 2 finds an optimal solution for more instances than Algorithm 3.

It is also clear that, for both algorithms, performance degrades mildly with n, and, perhaps surprisingly, performance for instances with either $d = 2$ or $d = 300$ is better than for instances with $d = 100$.

One interesting observation concerns instances of Class 2 (tables 4-6); these are much easier than the instances of the other classes. Not only are the running times small, even the instances with $n = 20$ and $d = 300$ are solved within 1000 seconds by the MIP (an exception is the running time of Algorithm 2 for $U(0, 100)$, $n = 20$, $d = 300$), also the quality of the solutions of the approximation algorithms is remarkably good: in particular, Algorithm 2 finds for the instances with mean 50 solutions within 1.2% of the optimum; for instances with mean 500 this percentage becomes 5%. A possible explanation for this phenomenon can be found in the component for which the maximum imbalance is attained (arg_k): due to the way these instances were generated, it is not surprising to see that this value approaches d. The same phenomenon, to a lesser extent, is present in the instances of Class 3 (Tables 7 and 8).

We summarize our findings as follows:

- Solving large instances ($n > 20$, $d > 100$) of the robust balanced assignment problem exactly, using a mixed integer programming formulation is a challenge.
- The approximation algorithms offer a trade-off in terms of quality of solution found, and running time needed; in particular, Algorithm 3 quickly finds solutions of reasonable quality.
- Structure present in the instances helps, both in terms of quality and running time, as witnessed by the instances of Class 2.

Table 1: Results Class 1, Uniform distributions

		U(0,100)			U(0,1000)		
		MIP	Alg2	Alg 3	MIP	Alg2	Alg3
n=10 d=2	time $\bar{x}$(s)	0.08 (0.02)	0.56 (0.21)	0.59 (0.03)	0.07 (0.01)	0.53 (0.15)	0.59 (0.03)
	Sol $\bar{x}$	38.4	38.7	39.4	393.7	395.2	409.9
	arg_k $\bar{x}$	1.65	1.55	1.45	1.4	1.3	1.4
	gap(%) $\bar{x}$ (s;#0)		0.7 (2.09 ; 9)	2.53 (3.29 ; 6)		0.36 (0.7 ; 7)	3.81 (4.39 ; 5)
n=10 d=100	time $\bar{x}$(s)	6.58 (1.3)	11.12 (1.02)	1.03 (0.03)	12.01 (2.83)	11.07 (0.46)	1.39 (0.04)
	Sol $\bar{x}$	94.1	95.3	96.7	936.2	947.6	959.3
	arg_k $\bar{x}$	41.43	57.67	46.15	66.15	52.23	61.7
	gap(%) $\bar{x}$ (s;#0)		1.28 (0.8 ; 2)	2.77 (0.72 ; 0)		1.22 (0.51 ; 0)	2.47 (0.64 ; 0)
n=10 d=300	time $\bar{x}$(s)	49.12 (9.76)	24.19 (1.25)	1.93 (0.06)	66.5 (8.11)	24.61 (0.96)	2.8 (0.05)
	Sol $\bar{x}$	97.4	98	98.8	964.6	972.4	979.2
	arg_k $\bar{x}$	213	157.1	148.7	153.4	192	156.4
	gap(%) $\bar{x}$ (s;#0)		0.62 (0.51 ; 4)	1.44 (0.69 ; 1)		0.81 (0.3 ; 0)	1.51 (0.26 ; 0)
n=20 d=2	time $\bar{x}$(s)	0.34 (0.09)	16.69 (4.62)	6.71 (0.37)	0.4 (0.09)	11.81 (2.36)	7.73 (0.34)
	Sol $\bar{x}$	31.4	32.1	33.7	312.3	315.1	343.3
	arg_k $\bar{x}$	1.3	1.35	1.45	1.45	1.55	1.5
	gap(%) $\bar{x}$ (s;#0)		2.23 (2.05 ; 4)	7.38 (6.53 ; 3)		0.92 (1.78 ; 6)	9.98 (4.21 ; 0)
n=20 d=100	time $\bar{x}$(s)	*9	793.4 (3.1)	13.21 (0.11)	*2	847.5 (16.65)	21.74 (0.74)
	Sol $\bar{x}$	95	98	98.7	942	967.9	977.4
	arg_k $\bar{x}$	52.36	46.11	52.07	46.03	56.6	42.8
	gap(%) $\bar{x}$ (s;#0)		3.16 (0 ; 0)	3.89 (0.48 ; 0)		2.75 (0.3 ; 0)	3.76 (0.24 ; 0)
n=20 d=300	time $\bar{x}$(s)	*0	1623 (14.74)	26.53 (0.45)	*0	1885.3 (30.31)	48.09 (0.51)
	Sol $\bar{x}$	98.3	99	99.8	978.1	984.3	988.9
	arg_k $\bar{x}$	142.6	143.8	153	160.9	164.5	175.7
	gap(%) $\bar{x}$ (s;#0)		0.71 (0.47 ; 3)	1.53 (0.69 ; 1)		0.64 (0.51 ; 1)	1.11 (0.53 ; 0)
*number of solved instances within 3600 seconds.							

Table 2: Results Class 1, Normal distributions with mean 50

		N(50,1)			N(50,10)		
		MIP	Alg2	Alg3	MIP	Alg2	Alg3
n=10 d=2	time $\bar{x}$(s)	0.1 (0.03)	0.4 (0.19)	0.56 (0.05)	0.09 (0.03)	0.45 (0.23)	0.54 (0.05)
	Sol $\bar{x}$	1.03	1.04	1.1	11.03	11.05	11.61
	arg_k $\bar{x}$	1.6	1.4	1.7	1.7	1.7	1.6
	gap(%) $\bar{x}$ (s;#0)		0.21 (0.63 ; 9)	6.63 (8.79 ; 4)		0.1 (0.29 ; 9)	5.84 (8.47 ; 3)
n=10 d=100	time $\bar{x}$(s)	8.98 (1.19)	9.66 (1.08)	1.3 (0.02)	8.97 (1.94)	11.15 (1.22)	1.31 (0.03)
	Sol $\bar{x}$	3.93	4.05	4.32	39.69	41.61	43.28
	arg_k $\bar{x}$	48.6	60.4	55.3	39	51.1	54
	gap(%) $\bar{x}$ (s;#0)		3.09 (1.15 ; 0)	9.87 (2.36 ; 0)		4.85 (2.13 ; 0)	9.05 (1.61 ; 0)
n=10 d=300	time $\bar{x}$(s)	73.34 (7.53)	26.72 (2.85)	2.65 (0.05)	59.81 (6.64)	24.77 (1.97)	2.66 (0.05)
	Sol $\bar{x}$	4.48	4.68	4.9	44.9	46.33	48.42
	arg_k $\bar{x}$	173.4	102.8	89.6	183.5	172.4	151.7
	gap(%) $\bar{x}$ (s;#0)		4.38 (0.81 ; 0)	9.32 (1.88 ; 0)		3.18 (1.09 ; 0)	7.83 (2.33 ; 0)
n=20 d=2	time $\bar{x}$(s)	0.46 (0.1)	15.02 (5.15)	7.05 (0.24)	0.49 (0.06)	11.25 (4.05)	7.3 (0.45)
	Sol $\bar{x}$	0.94	0.96	1.01	9.03	9.22	9.77
	arg_k $\bar{x}$	1.3	1.5	1.7	1.7	1.7	1.4
	gap(%) $\bar{x}$ (s;#0)		2.21 (2.06 ; 4)	7.58 (5.44 ; 0)		2.07 (1.75 ; 2)	8.6 (7.68 ; 2)
n=20 d=100	time $\bar{x}$(s)	*2	876.1 (46.63)	21.44 (0.17)	*8	873.6 (29.07)	21.95 (0.54)
	Sol $\bar{x}$	4	4.43	4.7	40.38	44.59	47.04
	arg_k $\bar{x}$	51.9	61.7	62.1	27.5	52.7	48.2
	gap(%) $\bar{x}$ (s;#0)		10.61 (1.64 ; 0)	17.36 (0.89 ; 0)		10.44 (1.05 ; 0)	16.49 (1.7 ; 0)
n=20 d=300	time $\bar{x}$(s)	*0	2088.4 (43.04)	50.85 (0.22)	*0	2091.7 (33.39)	51.49 (1.27)
	Sol $\bar{x}$	4.67	5.05	5.17	46.8	50.17	51.62
	arg_k $\bar{x}$	180.1	174.1	178.1	147.6	129.9	158.7
	gap(%) $\bar{x}$ (s;#0)		8.2 (1.26 ; 0)	10.74 (1.88 ; 0)		7.2 (0.89 ; 0)	10.32 (1.51 ; 0)
*number of solved instances within 3600 seconds.							

Table 3: Results Class 1, Normal distributions with mean 500

		N(500,1)			N(500,10)		
		MIP	Alg2	Alg3	MIP	Alg2	Alg3
n=10 d=2	time $\bar{x}$(s)	0.11 (0.03)	0.46 (0.17)	0.65 (0.05)	0.1 (0.03)	0.52 (0.14)	0.66 (0.07)
	Sol $\bar{x}$	1.06	1.07	1.1	11.35	11.42	11.83
	arg_k $\bar{x}$	1.4	1.5	1.4	1.9	1.7	1.4
	gap(%) $\bar{x}$ (s;#0)		0.99 (2.09 ; 8)	4.2 (8.51 ; 7)		0.56 (1.33 ; 8)	4.41 (5.3 ; 3)
n=10 d=100	time $\bar{x}$(s)	8.7 (1.95)	11.09 (0.95)	1.63 (0.07)	8.32 (1.58)	11.38 (1.31)	1.85 (0.11)
	Sol $\bar{x}$	3.96	4.13	4.32	39.69	41.42	43.5
	arg_k $\bar{x}$	60	57.3	40.1	51.6	58.3	43.3
	gap(%) $\bar{x}$ (s;#0)		4.32 (2.09 ; 0)	9.14 (3.54 ; 0)		4.36 (2.3 ; 0)	9.6 (2.88 ; 0)
n=10 d=300	time $\bar{x}$(s)	69.12 (8.54)	25.57 (1.53)	3.03 (0.06)	69.27 (7.7)	26.11 (1.26)	3.59 (0.19)
	Sol $\bar{x}$	4.47	4.65	4.84	45.25	47.06	48.44
	arg_k $\bar{x}$	149	149.4	112.2	134.3	196.9	186.4
	gap(%) $\bar{x}$ (s;#0)		3.99 (1.91 ; 0)	8.17 (3.2 ; 0)		4.01 (1.29 ; 0)	7.05 (2.42 ; 0)
n=20 d=2	time $\bar{x}$(s)	0.51 (0.17)	12.56 (3.95)	8.98 (0.71)	0.46 (0.09)	10.23 (2.05)	9.13 (0.35)
	Sol $\bar{x}$	0.91	0.93	0.99	8.72	8.84	9.2
	arg_k $\bar{x}$	1.5	1.4	1.4	1.4	1.4	1.3
	gap(%) $\bar{x}$ (s;#0)		2.75 (1.73 ; 1)	9.78 (5.73 ; 0)		1.37 (2.96 ; 6)	5.43 (7.07 ; 3)
n=20 d=100	time $\bar{x}$(s)	*8	881.4 (18.14)	29.1 (2.55)	*9	884.4 (21.97)	29.71 (0.35)
	Sol $\bar{x}$	4.02	4.47	4.71	40.1	44.68	46.74
	arg_k $\bar{x}$	69.2	35.9	58.7	46.8	47	56.1
	gap(%) $\bar{x}$ (s;#0)		11.22 (1.14 ; 0)	17.4 (1.32 ; 0)		11.41 (0.9 ; 0)	16.56 (1.66 ; 0)
n=20 d=300	time $\bar{x}$(s)	*0	2079.4 (25.34)	73.12 (2.08)	*0	2062.7 (38.51)	77.28 (4.29)
	Sol $\bar{x}$	4.69	5.02	5.26	46.86	50.44	52.14
	arg_k $\bar{x}$	124.6	163.7	153.6	181.6	97.5	142.4
	gap(%) $\bar{x}$ (s;#0)		7.07 (0.87 ; 0)	12 (0.95 ; 0)		7.65 (0.89 ; 0)	11.29 (1.57 ; 0)
*number of solved instances within 3600 seconds.							

Table 4: Results Class 2, Uniform distributions

		U(0,100)			U(0,1000)		
		MIP	Alg2	Alg3	MIP	Alg2	Alg3
n=10 d=2	time $\bar{x}$(s)	0.09 (0.03)	0.36 (0.12)	0.59 (0.06)	0.07 (0.01)	0.3 (0.11)	0.61 (0.06)
	Sol $\bar{x}$	29.9	30.2	30.4	274.2	275.1	279.2
	arg_k $\bar{x}$	1.4	1.3	1.4	1.5	1.6	1.4
	gap(%) $\bar{x}$ (s;#0)		0.95 (2.09 ; 8)	1.68 (1.7 ; 5)		0.28 (0.66 ; 8)	1.73 (3.73 ; 8)
n=10 d=100	time $\bar{x}$(s)	2.39 (0.44)	5.29 (2.79)	0.95 (0.05)	4.57 (1.43)	2.15 (0.42)	1.32 (0.08)
	Sol $\bar{x}$	98.5	98.5	98.5	977.1	977.1	978.2
	arg_k $\bar{x}$	97.75	98.25	98.43	99.05	98.75	98.75
	gap(%) $\bar{x}$ (s;#0)		0 (0 ; 10)	0 (0 ; 10)		0 (0 ; 10)	0.11 (0.17 ; 6)
n=10 d=300	time $\bar{x}$(s)	10.31 (3.36)	39.39 (0.35)	1.17 (0.03)	34.46 (8.9)	6.49 (0.84)	2.39 (0.08)
	Sol $\bar{x}$	100	100	100	993	993	993.1
	arg_k $\bar{x}$	296.1	296.6	296.6	298.6	298.5	298.5
	gap(%) $\bar{x}$ (s;#0)		0 (0 ; 10)	0 (0 ; 10)		0 (0 ; 10)	0.01 (0.03 ; 9)
n=20 d=2	time $\bar{x}$(s)	0.29 (0.08)	12.67 (3.75)	7.39 (0.36)	0.4 (0.12)	10.06 (2.15)	7.67 (0.47)
	Sol $\bar{x}$	22.8	23	24.1	233.6	236.6	253.9
	arg_k $\bar{x}$	1.45	1.55	1.75	1.4	1.3	1.5
	gap(%) $\bar{x}$ (s;#0)		0.85 (1.7 ; 8)	5.71 (5.65 ; 3)		1.33 (1.91 ; 5)	8.65 (5.88 ; 0)
n=20 d=100	time $\bar{x}$(s)	90.36 (39.9)	147.1 (5.48)	12.63 (0.44)	181.5 (61.55)	94.59 (11.17)	19.21 (0.53)
	Sol $\bar{x}$	98	98	98	972.4	972.4	973.5
	arg_k $\bar{x}$	97.82	97.51	97.48	98.25	98.3	97.95
	gap(%) $\bar{x}$ (s;#0)		0 (0 ; 10)	0 (0 ; 10)		0 (0 ; 10)	0.11 (0.14 ; 5)
n=20 d=300	time $\bar{x}$(s)	84.71 (84.34)	2176.3 (13.85)	17.02 (0.38)	855.6 (324.4)	289.2 (49.34)	37.43 (0.53)
	Sol $\bar{x}$	100	100	100	990.9	991	991.4
	arg_k $\bar{x}$	295.9	296	296	297.8	298.2	297.8
	gap(%) $\bar{x}$ (s;#0)		0 (0 ; 10)	0 (0 ; 10)		0.01 (0.03 ; 9)	0.05 (0.05 ; 5)
*number of solved instances within 3600 seconds.							

Table 5: Results Class 2, Normal distributions with mean 50

		N(50,1)			N(50,10)		
		MIP	Alg2	Alg3	MIP	Alg2	Alg3
n=10 d=2	time $\bar{x}$(s)	0.1 (0.02)	0.34 (0.12)	0.55 (0.04)	0.12 (0.02)	0.44 (0.13)	0.57 (0.04)
	Sol $\bar{x}$	0.94	0.94	0.97	9.63	9.74	10
	arg_k $\bar{x}$	1.6	1.6	1.4	1.2	1.5	1.4
	gap(%) $\bar{x}$ (s;#0)		0.06 (0.18 ; 9)	2.62 (6.13 ; 8)		1.18 (2.87 ; 6)	4.01 (6.77 ; 5)
n=10 d=100	time $\bar{x}$(s)	5.89 (1.36)	2.82 (0.45)	1.34 (0.06)	6.63 (3.38)	2.77 (0.48)	1.31 (0.05)
	Sol $\bar{x}$	4.6	4.6	4.65	46.24	46.25	46.62
	arg_k $\bar{x}$	98.9	98.8	98.9	98.5	98.6	98.5
	gap(%) $\bar{x}$ (s;#0)		0 (0.01 ; 8)	1.23 (1.94 ; 5)		0.02 (0.06 ; 4)	0.8 (1.33 ; 4)
n=10 d=300	time $\bar{x}$(s)	49.03 (9.68)	7.71 (1.72)	2.7 (0.08)	44.37 (31.66)	8.45 (1.86)	2.73 (0.1)
	Sol $\bar{x}$	5.4	5.41	5.43	54.35	54.39	54.7
	arg_k $\bar{x}$	298.5	298.7	298.8	298.8	298.8	298.7
	gap(%) $\bar{x}$ (s;#0)		0.08 (0.17 ; 8)	0.61 (0.8 ; 5)		0.06 (0.17 ; 5)	0.64 (0.67 ; 1)
n=20 d=2	time $\bar{x}$(s)	0.44 (0.13)	9.38 (2.15)	7.13 (0.44)	0.41 (0.09)	7.9 (1.84)	7.03 (0.29)
	Sol $\bar{x}$	0.71	0.72	0.76	7.1	7.18	7.77
	arg_k $\bar{x}$	1.3	1.3	1.5	1.7	1.6	1.6
	gap(%) $\bar{x}$ (s;#0)		0.98 (1.24 ; 6)	7.65 (5.22 ; 1)		1.03 (1.94 ; 7)	9.38 (5.44 ; 0)
n=20 d=100	time $\bar{x}$(s)	146.1 (54.96)	108.7 (18.25)	22.56 (0.9)	145.7 (37.07)	106.5 (8.68)	22.34 (0.3)
	Sol $\bar{x}$	4.39	4.39	4.52	44.15	44.21	45.23
	arg_k $\bar{x}$	98.3	98.4	98.3	98.4	98.4	98
	gap(%) $\bar{x}$ (s;#0)		0.14 (0.33 ; 8)	3.04 (1.84 ; 0)		0.14 (0.28 ; 3)	2.46 (1.39 ; 1)
n=20 d=300	time $\bar{x}$(s)	880.8 (138.2)	325.5 (36.74)	53.04 (0.67)	660.1 (99.57)	320.8 (53.01)	53 (1.15)
	Sol $\bar{x}$	5.22	5.22	5.33	51.98	52.02	53.12
	arg_k $\bar{x}$	298.3	298.3	298.5	298.2	298.1	298.3
	gap(%) $\bar{x}$ (s;#0)		0.09 (0.21 ; 8)	2.1 (0.82 ; 1)		0.09 (0.18 ; 1)	2.19 (1.46 ; 0)
*number of solved instances within 3600 seconds.							

Table 6: Results Class 2, Normal distributions with mean 500

		N(500,1)			N(500,10)		
		MIP	Alg2	Alg3	MIP	Alg2	Alg3
n=10 d=2	time $\bar{x}$(s)	0.1 (0.03)	0.36 (0.16)	0.58 (0.04)	0.12 (0.04)	0.39 (0.12)	0.54 (0.03)
	Sol $\bar{x}$	0.86	0.89	0.91	9.49	9.52	9.81
	arg_k $\bar{x}$	1.8	1.6	1.5	1.4	1.4	1.3
	gap(%) $\bar{x}$ (s;#0)		2.63 (2.99 ; 4)	5.93 (4.87 ; 3)		0.29 (0.88 ; 9)	3.58 (5.95 ; 5)
n=10 d=100	time $\bar{x}$(s)	4.54 (1.05)	2.48 (0.59)	1.35 (0.06)	6.45 (4.15)	2.94 (0.52)	1.3 (0.07)
	Sol $\bar{x}$	4.55	4.55	4.58	46.68	46.77	47.38
	arg_k $\bar{x}$	98.5	98.5	98.4	98.8	98.8	98.8
	gap(%) $\bar{x}$ (s;#0)		0 (0 ; 10)	0.76 (1.18 ; 7)		0.2 (0.6 ; 6)	1.5 (1.74 ; 2)
n=10 d=300	time $\bar{x}$(s)	39.62 (8.05)	7.78 (1.35)	2.77 (0.09)	46 (25.01)	8.99 (1.17)	2.69 (0.09)
	Sol $\bar{x}$	5.37	5.37	5.4	55.07	55.11	55.54
	arg_k $\bar{x}$	298.5	298.6	298.4	298.9	298.9	298.9
	gap(%) $\bar{x}$ (s;#0)		0.01 (0.03 ; 8)	0.48 (0.57 ; 5)		0.07 (0.12 ; 5)	0.83 (0.82 ; 2)
n=20 d=2	time $\bar{x}$(s)	0.45 (0.08)	9.41 (3.35)	6.93 (0.29)	0.47 (0.13)	8.82 (3.12)	7.09 (0.22)
	Sol $\bar{x}$	0.71	0.74	0.76	6.95	7.09	7.27
	arg_k $\bar{x}$	1.4	1.4	1.5	1.5	1.7	1.4
	gap(%) $\bar{x}$ (s;#0)		3.9 (2.08 ; 1)	6.74 (8.08 ; 4)		2 (2.53 ; 3)	4.88 (5.45 ; 2)
n=20 d=100	time $\bar{x}$(s)	153 (27.36)	105.6 (17.56)	21.98 (0.54)	116.5 (39.76)	100.8 (16.18)	22.3 (0.33)
	Sol $\bar{x}$	4.39	4.4	4.53	43.78	43.79	45.04
	arg_k $\bar{x}$	98.2	98.2	97.8	98.5	98.5	98.2
	gap(%) $\bar{x}$ (s;#0)		0.02 (0.05 ; 9)	3.06 (1.55 ; 0)		0.02 (0.05 ; 8)	2.87 (1.2 ; 0)
n=20 d=300	time $\bar{x}$(s)	731.5 (195.6)	309.3 (34.39)	52.92 (0.36)	630.8 (86.27)	331.8 (29.13)	52.69 (0.54)
	Sol $\bar{x}$	5.19	5.2	5.31	52.29	52.32	53.13
	arg_k $\bar{x}$	298.3	298.3	297.9	298.4	298.4	298.1
	gap(%) $\bar{x}$ (s;#0)		0.03 (0.06 ; 8)	2.2 (1.09 ; 0)		0.06 (0.08 ; 5)	1.61 (0.94 ; 1)
*number of solved instances within 3600 seconds.							

Table 7: Results Class 3, Uniform distributions

		U(0,100)			U(0,1000)		
		MIP	Alg2	Alg3	MIP	Alg2	Alg3
n=10 d=2	time $\bar{x}$(s)	0.09 (0.02)	0.48 (0.16)	0.6 (0.03)	0.09 (0.02)	0.52 (0.21)	0.6 (0.06)
	Sol $\bar{x}$	27.3	27.3	28.6	278.3	279.8	289.2
	arg$_k$ $\bar{x}$	1.5	1.65	1.6	1.7	1.7	1.5
	gap(%) $\bar{x}$ (s;#0)		0 (0 ; 10)	4.8 (4.51 ; 3)		0.49 (0.85 ; 7)	3.89 (5.07 ; 5)
n=10 d=100	time $\bar{x}$(s)	5.76 (1.55)	7.99 (1.14)	1.08 (0.03)	6.61 (1.59)	7.37 (0.97)	1.35 (0.03)
	Sol $\bar{x}$	76.2	78.5	80.4	761.8	774.2	803.2
	arg$_k$ $\bar{x}$	86.68	91.95	92.62	87.8	88.6	94.2
	gap(%) $\bar{x}$ (s;#0)		3.02 (1.57 ; 1)	5.51 (2.4 ; 1)		1.63 (1.53 ; 3)	5.44 (2.24 ; 0)
n=10 d=300	time $\bar{x}$(s)	40.18 (7.48)	17.04 (1.59)	1.94 (0.04)	52.17 (10.34)	17.93 (2)	2.61 (0.05)
	Sol $\bar{x}$	83.4	85	87.9	837.6	848.8	872.4
	arg$_k$ $\bar{x}$	282.7	281.3	286.4	279	281.8	287.8
	gap(%) $\bar{x}$ (s;#0)		1.92 (1.23 ; 2)	5.4 (1.46 ; 0)		1.34 (0.89 ; 0)	4.16 (0.96 ; 0)
n=20 d=2	time $\bar{x}$(s)	0.35 (0.1)	16.48 (5.11)	7.11 (0.2)	0.4 (0.11)	12.2 (3.64)	7.46 (0.48)
	Sol $\bar{x}$	22.2	22.3	24	222	223.7	232.8
	arg$_k$ $\bar{x}$	1.6	1.55	1.55	1.5	1.55	1.55
	gap(%) $\bar{x}$ (s;#0)		0.43 (1.3 ; 9)	8.22 (6.27 ; 2)		0.76 (0.75 ; 4)	4.83 (4.98 ; 2)
n=20 d=100	time $\bar{x}$(s)	1155.9 (270.6)	572 (14.76)	14.01 (0.25)	906.5 (157.8)	599.5 (24.8)	21.46 (0.84)
	Sol $\bar{x}$	76.1	80.9	85	758.3	801.3	835.3
	arg$_k$ $\bar{x}$	90.37	89.63	94.17	87.45	90.6	92
	gap(%) $\bar{x}$ (s;#0)		6.31 (0.52 ; 0)	11.7 (0.94 ; 0)		5.68 (1.2 ; 0)	10.15 (0.91 ; 0)
n=20 d=300	time $\bar{x}$(s)	*0	1347.3 (27.52)	29.57 (0.37)	*0	1521.4 (35.42)	48.55 (1.41)
	Sol $\bar{x}$	85.4	88.2	90.6	850.5	881	903.8
	arg$_k$ $\bar{x}$	282.1	287.6	289.9	279.5	285.1	287.4
	gap(%) $\bar{x}$ (s;#0)		3.29 (1.17 ; 0)	6.09 (0.75 ; 0)		3.59 (0.94 ; 0)	6.29 (1.9 ; 0)
*number of solved instances within 3600 seconds.							

Table 8: Results Class 3, Normal distributions

		N(50,.)			N(500,.)		
		MIP	Alg2	Alg3	MIP	Alg2	Alg3
n=10 d=2	time $\bar{x}$(s)	0.11 (0.03)	0.41 (0.12)	0.66 (0.07)	0.1 (0.02)	0.48 (0.18)	0.7 (0.09)
	Sol $\bar{x}$	12.72	12.87	13.62	138.4	142.5	144.9
	arg$_k$ $\bar{x}$	1.3	1.3	1.3	1.6	1.6	1.7
	gap(%) $\bar{x}$ (s;#0)		1.3 (2.9 ; 7)	7.34 (7.82 ; 3)		2.87 (4.85 ; 0)	4.6 (5.29 ; 0)
n=10 d=100	time $\bar{x}$(s)	6.95 (1.16)	8.48 (1.08)	1.98 (0.17)	6.54 (1.16)	8.7 (1.12)	1.9 (0.13)
	Sol $\bar{x}$	47.84	49.2	52.93	477.3	497	518.9
	arg$_k$ $\bar{x}$	86.6	91.9	79.2	83.8	83.9	82.3
	gap(%) $\bar{x}$ (s;#0)		2.86 (2.46 ; 1)	10.64 (3.41 ; 0)		4.13 (2.32 ; 0)	8.72 (4.01 ; 0)
n=10 d=300	time $\bar{x}$(s)	56.7 (5.17)	23.29 (1.56)	4.41 (0.83)	62.77 (11.21)	21.77 (1.79)	3.72 (0.18)
	Sol $\bar{x}$	57.39	59.69	63.26	565.6	585.1	616.5
	arg$_k$ $\bar{x}$	271.5	282.7	265.5	264.5	266	270.6
	gap(%) $\bar{x}$ (s;#0)		4 (1.41 ; 0)	10.23 (2.47 ; 0)		3.44 (2.31 ; 0)	9.01 (2.05 ; 0)
n=20 d=2	time $\bar{x}$(s)	0.43 (0.08)	10.96 (2.43)	9.5 (0.37)	0.47 (0.06)	9.52 (3.3)	9.25 (0.33)
	Sol $\bar{x}$	10.88	11.01	11.97	104.1	105.7	110.6
	arg$_k$ $\bar{x}$	1.6	1.6	1.7	1.8	1.4	1.5
	gap(%) $\bar{x}$ (s;#0)		1.14 (1.37 ; 4)	9.76 (6.62 ; 0)		1.56 (1.76 ; 1)	6.46 (7.41 ; 2)
n=20 d=100	time $\bar{x}$(s)	741.2 (160.6)	731.2 (41.95)	30.97 (1.77)	824 (175.9)	737.7 (19.69)	29.97 (0.4)
	Sol $\bar{x}$	47.5	53.3	57.27	475.1	537	570.3
	arg$_k$ $\bar{x}$	82.8	79.6	90.1	88.1	87.4	89.5
	gap(%) $\bar{x}$ (s;#0)		12.2 (2.21 ; 0)	20.58 (1.87 ; 0)		13.03 (1.38 ; 0)	20.04 (1.85 ; 0)
n=20 d=300	time $\bar{x}$(s)	*1	1895.6 (24.33)	70.99 (0.34)	*0	1766.8 (152.3)	71.32 (0.36)
	Sol $\bar{x}$	58.37	64.09	67.46	588.4	641.8	683.7
	arg$_k$ $\bar{x}$	275.2	267.5	267.4	276.1	269.9	272.4
	gap(%) $\bar{x}$ (s;#0)		9.82 (1.17 ; 0)	15.6 (3.03 ; 0)		9.08 (1.51 ; 0)	16.21 (1.66 ; 0)
*number of solved instances within 3600 seconds.							

2.7 CONCLUSION

We introduce balanced optimization problems with vector costs, and show its equivalence to robust balanced optimization problems. We propose a framework that generalizes the one introduced by Martello et al. (1984). We provide a polynomial time algorithm when the dimension d is fixed, and we describe two 2-approximation algorithms for each problem in our framework. Further, we give results for a number of problems in the framework: we settle their complexity, and for many of them the existence of a polynomial time $(2-\epsilon)$-approximation algorithm implies P=NP. Finally, we provide computational evidence for the quality of the approximation algorithms applied to the robust balanced assignment problem.

3

THE TRANSPORTATION PROBLEM WITH CONFLICTS

> Theorems are fun especially when you are the prover.
>
> *Carl Pomerance*

3.1 INTRODUCTION

The transportation problem is a fundamental problem in Operations Research, where items need to be transported from supply nodes (each with a given supply) to demand nodes (each with a given demand) in the cheapest possible way. In this chapter we introduce a generalization of the transportation problem where, each supply node has a (possibly empty) set of conflicting pairs of demand nodes, and each demand node a (possibly empty) set of conflicting pairs of supply nodes. In any feasible solution a supply node may not send supply to both demand nodes of any of its conflicting pair. Likewise, a demand node may not receive supply from both supply nodes of any of its conflicting pairs. We call the resulting problem the transportation problem with conflicts (TPC).

3.1.1 *Problem Statement*

In the Transportation Problem with Conflicts (TPC) we are given a complete bipartite graph $(S \cup D, E)$, where (see also Figure 6):

S: is the set of supply nodes, with for each $i \in S$,

This chapter is the result of a collaboration with F. C. R. Spieksma and G. J. Woeginger. It corresponds to Research Report KBI 1719, and has been submitted for publication.

- a supply $s_i \in \mathbb{N}$, and
- a conflict set C_i containing forbidden pairs of demand nodes,

D: is the set of demand nodes, with for each $j \in D$,

- a demand $d_j \in \mathbb{N}$, and
- a conflict set F_j containing forbidden pairs of supply nodes,

E: is the complete edge set, with for each $i \in S$ and $j \in D$ an edge $(i, j) \in E$ for which we have,

- a capacity $u_{i,j} \in \mathbb{N}$, and
- a weight $w_{i,j} \in \mathbb{N}$.

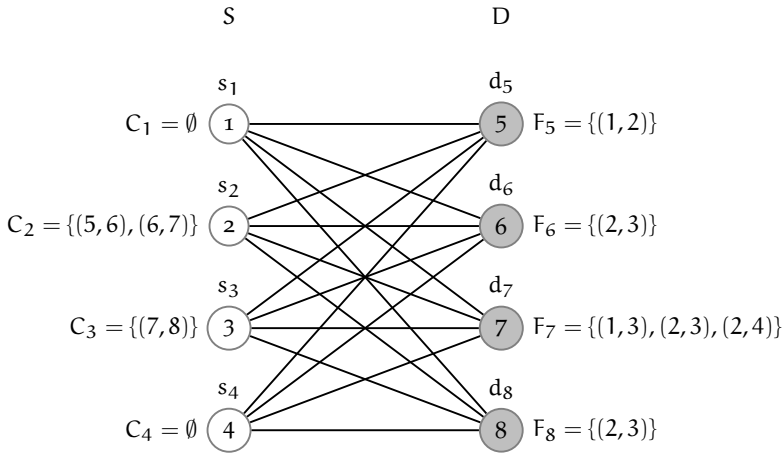


Figure 6: Transportation Problem with Conflicts

Following the literature on the classical Transportation Problem, we assume that $\sum_{i \in S} s_i = \sum_{j \in D} d_j$. A solution is an integral vector x, with $x_{i,j} \in \mathbb{N}$ indicating for each edge $(i, j) \in E$, how much supply is sent from supply node $i \in S$ to demand node $j \in D$. The value of a solution equals $\sum_{i \in S} \sum_{j \in D} w_{i,j} \cdot x_{i,j}$.

Given a solution x, we say that a conflict occurs if $x_{i,j_1} > 0$ and $x_{i,j_2} > 0$, while $(j_1, j_2) \in C_i$ for some $i \in S$. Likewise, a conflict occurs if $x_{i_1,j} > 0$ and $x_{i_2,j} > 0$, while $(i_1, i_2) \in F_j$ for some $j \in D$. A solution x is not feasible when a conflict occurs or when $x_{i,j} > u_{i,j}$ for an edge $(i, j) \in E$.

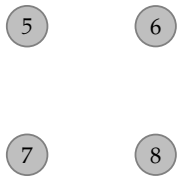
We say that demand (supply) constraints are **fulfilled** if for all $j \in D$, $\sum_{i \in S} x_{i,j} = d_j$ (for all $i \in S$, $\sum_{j \in S} x_{i,j} = s_i$). We say that demand (supply) constraints are **respected** if for all $j \in D$, $\sum_{i \in S} x_{i,j} \leqslant d_j$ (for all $i \in S$, $\sum_{j \in S} x_{i,j} \leqslant s_i$).

We consider two problems:

(I) find a feasible solution fulfilling all demand and supply constraints, while minimizing the value of a solution (min-TPC), and
(II) find a feasible solution respecting all demand and supply constraints, while maximizing the value of a solution (max-TPC).

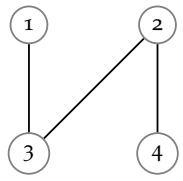
3.1.2 *Terminology and Notation*

An important tool in our analysis are conflict graphs. We build an induced *conflict graph* G_{F_j} for each demand node $j \in D$ as follows: there is a vertex for each supply node and two vertices are connected if and only if the corresponding supply nodes constitute a forbidden pair in F_j. A similar procedure is used to build an induced conflict graph G_{C_i} for each supply node $i \in S$. For an example, we refer to Figures 7 & 8 for the conflict graphs of C_1 and F_7 from Figure 6.



$C_1 = \emptyset$

Figure 7: Conflict Graph G_{C_1}



$F_7 = \{(1,3), (2,3), (2,4)\}$

Figure 8: Conflict Graph G_{F_7}

Clearly, in case of identical conflict sets F, the induced conflict graphs are identical as well; we then refer to *the* conflict graph G_F.

The use of conflict graphs allows us to easily express special structures occurring in the conflict sets. For example, consider the previously mentioned Red-Blue Transportation problem (Section 1.2.2), in which we need to assign male and female patients to hospital rooms. In the Red-Blue Transportation problem we have an identical conflict set F that contains every pair consisting of a male and a female patient. Equivalently, the conflict graph induced by F is complete bipartite. Indeed, in the conflict graph there is an edge between every red supply node (female patient) and every blue supply node (male patient), see also Figure 9.

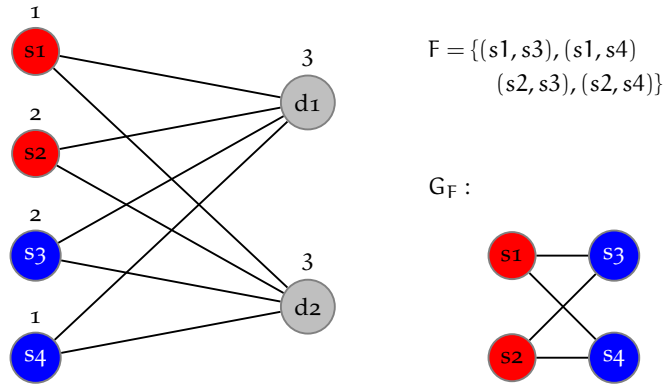


Figure 9: Red-Blue Transportation

Throughout this chapter we will discuss several special cases of TPC. We use a three-field notation to systematize the different special cases of TPC: TPC(α, β, χ).

Definition 3.1. *TPC(α, β, χ) is a special case of TPC where;*

$\alpha \in \{C_i, C, C^B, \emptyset\}$ *describes the nature of the conflict sets of the supply nodes.*

C_i: *arbitrary conflict sets, each supply node $i \in S$ has a (possibly empty) conflict set.*

C: *identical conflict sets, each supply node* $i \in S$ *has conflict set* C.

C^B: *identical conflict sets, and its induced conflict graph* G_C *is complete bipartite.*

$\emptyset$: *no conflict between demand nodes, each supply node* $i \in S$ *has an empty conflict set.*

$\beta \in \{F_j, F, F^B, \emptyset\}$ *describes the nature of the conflict sets of the demand nodes.*

F_j: *arbitrary conflict sets, each demand node* $j \in D$ *has a (possibly empty) conflict set.*

F: *identical conflict sets, each demand node* $j \in D$ *has conflict set* F.

F^B: *identical conflict sets, and its induced conflict graph* G_F *is complete bipartite.*

$\emptyset$: *no conflict between supply nodes, each demand node* $j \in D$ *has an empty conflict set.*

$\chi \in \{u_{i,j}, \infty, 1\}$ *describes the nature of the edge capacities.*

$u_{i,j}$: *arbitrary capacities.*

∞: *uncapacitated, or equivalently* $u_{i,j} \geqslant \min\{s_i, d_j\}$

1: *then* $u_{i,j} = 1$, *for every edge* $(i,j) \in E$.

We call the TPC *one-sided* if either $C_i = \emptyset$ for all supply nodes $i \in S$, or $F_j = \emptyset$ for all demand nodes $j \in D$.

3.1.3 *Related Literature*

Special cases and applications

We first explicitly point out how the three examples mentioned in Section 1.2.2 are special cases of TPC.

CONFLICT-AWARE WEIGHTED BIPARTITE b-MATCHING:
Chen et al. (2016) describe the Conflict-aware weighted Bipartite b-matching problem (CA-WBM). As mentioned in Section 1.2.2, their motivation for research comes from E-commerce, where suppliers are companies that want to promote their products and demand

nodes are the potential customers. In their most general problem description there is, as input, a threshold t, the number of conflicts allowed in a feasible solution. The problem with $t = 0$, is a special case of max-TPC where we have edge capacities $u_{i,j} = 1$ for each $(i, j) \in E$, identical conflict sets $C_i = C$ for each supply node $i \in S$ and identical conflict set $F_j = F$ for each demand node $j \in D$, i.e. max-TPC$(C, F, 1)$.

They prove that their problem is NP-hard even if the problem is one-sided and no conflicts ($t = 0$) are allowed. Let d be the maximum degree of both conflict graphs G_C and G_F. They present a greedy $(2 + d)$-approximation for the one-sided case. The same analysis can be used to give a greedy $(2 + 2d)$-approximation algorithm for the two-sided case.

TRANSPORTATION PROBLEM WITH EXCLUSIONARY SIDE CONSTRAINTS:
Cao (1992) describes the Transportation Problem with Exclusionary Side Constraints (TPESC). As mentioned in Section 1.2.2, the motivation comes from storage management of containers. This problem is a special case of min-TPC where we have one-sided conflicts and no edge capacities, i.e. min-TPC$(\emptyset, F_j, \infty)$.

Cao (1992) and later Sun (2002) describe Branch and Bound approaches to solve the problem. Cao and Uebe (1995) and later Sun (1998) use Tabu-Search methods to solve the problem. Syarif and Gen (2003) describe a genetic algorithm. Goossens and Spieksma (2009) prove that this problem is strongly NP-hard even if the conflict sets are identical, i.e. $F_j = F$ for all $j \in D$. They also prove that no polynomial time constant factor approximation exists even if there is only one non-empty conflict set and $|D| = 2$ (unless P = NP).

RED-BLUE TRANSPORTATION PROBLEM:
Vancroonenburg et al. (2014) describe the Red-Blue Transportation

Problem (RBTP). As mentioned in Section 1.2.2, one of their motivations is the assignment of patients to hospital rooms. RBTP is defined for both the minimization as well as the maximization objective. This problem is a special case of TPC where we have a one-sided identical conflict set F and no edge capacities. Actually, the conflict set F induces a complete bipartite graph as conflict graph (see also Section 3.2), i.e. RBTP can be described as TPC$(\emptyset, F^B, \infty)$.

For the minimization objective Vancroonenburg et al. (2014) prove that no constant-factor approximation algorithm for RBTP can exist, even if $s_i = 1$ and $d_j = 3$ for all $i \in S$, $j \in D$ (unless P=NP). For the maximization objective the problem remains NP-hard and they present three different $\frac{1}{2}$-approximation algorithms.

Other related problems

MAX FLOW WITH DISJUNCTIVE CONSTRAINTS:

In the maximum flow problem with negative disjunctive constraints we are given a directed graph $G = (N, A)$, source $s \in N$, a sink $t \in N$, and there is a capacity $u_{i,j}$ for each arc $(i, j) \in A$. In addition there are negative disjunctive constraints, meaning that pairs of arcs are given, at most one of which is allowed to carry a positive amount of flow in a feasible solution. The goal is to maximize the amount of flow sent from s to t while respecting the capacities and the negative disjunctive constraints.

Similar to the construction sketched in Section 3.1.2 one can build a conflict graph in which there is a node for each arc (i, j) and there is an edge between two arcs if they are in conflict.

Pferschy and Schauer (2013) prove that this problem is NP-hard, even if the induced conflict graph consists only of unconnected edges and the directed graph $G = (N, A)$ only consists of disjoint paths of length three. Notice that this problem is more general than max-TPC due to the arbitrary graph structure.

MAXIMUM MATCHING WITH DISJUNCTIVE CONSTRAINTS:

Another well-known problem related to the classical Transportation problem is maximum matching. In the Maximum Matching problem with negative disjunctive constraint we are given a graph $G = (V, E)$ with edge weights and negative disjunctive constraints on the edges. The goal is to find a matching with maximum weight. Darmann et al. (2011) show that the maximum matching problem with negative disjunctive constraints is NP-hard even for conflict graphs where every connected component is a single edge.

Again, this result does not apply to TPC due to the fact that TPC is defined on a bipartite graph.

TWO-TO-ONE ASSIGNMENT PROBLEM:

Goossens et al. (2012) describe the two-to-one assignment problem (2-1-AP) as follows. We are given a set X of $2n$ elements and a set Y of n elements. A feasible triple cost $c_{i_1,i_2,j}$ is associated to two distinct elements $i_1, i_2 \in X$ and an element $j \in Y$; the goal is to select n feasible triples of minimum total cost such that each element occurs once in a selected triple.

They prove that the existence of a PTAS would imply P=NP and they provide a constant-factor approximation for the special case of decomposable costs satisfying the triangle inequality.

It is not difficult to see that min-TPC$(\emptyset, F, \infty)$ with $s_i = 1$ for all $i \in S$ and $d_j = 2$ for all $j \in D$ is a special case of 2-1-AP. Indeed, observe that any feasible solution of min-TPC$(\emptyset, F, \infty)$ with $d_j = 2$ also consists of triples, where a triple consists of a demand node and two supply nodes that are not in conflict. Given an instance of min-TPC, we create an instance of the 2-1-AP as follows. Let set X contain all supply nodes and set Y all demand nodes. Consider now a triple consisting of a node $j \in Y$ and two nodes $i_1, i_2 \in X$, that are not in conflict for demand node $j \in D$. We set its cost to be $c_{i_1,i_2,j} = w_{i_1,j} + w_{i_2,j}$; the cost of any other triple, where $i_1, i_2 \in X$

are in conflict for demand node j, is set to a large number. This shows that min-TPC is a special case of 2-1-AP. The costs we define however are not decomposable in the sense of Goossens et al. (2012).

3.2 A PREVIEW

Let us first informally argue why max-TPC in case of a single demand node is an NP-hard problem. Clearly, in any feasible solution, the set of supply nodes $i \in S$ that send flow to a particular demand node $j \in D$ (i.e. those $i \in S$ with $x_{i,j} > 0$) form an independent set in the induced conflict graph of that demand node $j \in D$. Indeed, since apparently none of the supply nodes assigned are in conflict, there is no edge connecting them in the conflict graph. This shows that Maximum Independent Set (MIS) is a special case of max-TPC. Indeed, given an instance of MIS, i.e. a graph $G = (V, E)$ with $|V| = n$, we can construct an instance of max-TPC with n supply nodes each with supply 1 and a single demand node with demand n whose induced conflict graph is equivalent to the graph G, finally we define the edge weights to be 1 (see also Figure 10).

This means that max-TPC with a single demand node is at least as difficult as the maximum independent set problem.

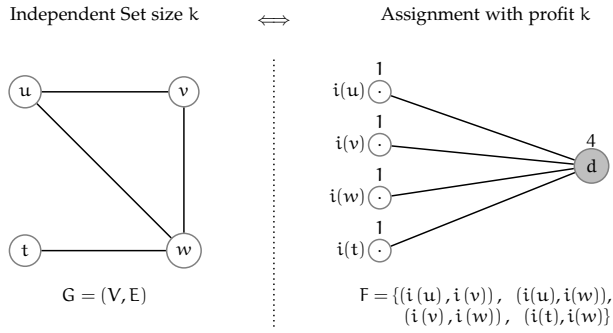


Figure 10: MIS and Max-TPC

Note that for min-TPC the above construction does not work, as min-TPC requires that all supply and demand constraints are fulfilled. In the above construction supply constraints can only be fulfilled if the graph G of the MIS instance contains no edges, i.e. no pair of supply nodes is in conflict. However, with a similar reduction, it is shown that no PTAS exists for min-TPC$(\emptyset, F_j, \infty)$, even with $|D| \geqslant 2$ and exactly one demand node having a non-empty conflict set (unless P=NP), (Goossens and Spieksma, 2009).

We now review the existing complexity results for TPC.

Goossens and Spieksma (2009) prove that:

(I) min-TPC$(\emptyset, F, \infty)$ is weakly NP-hard for $|D| = 2$,
(II) min-TPC$(\emptyset, F_j, \infty)$ is weakly NP-hard for each fixed $|S| \geqslant 2$, and
(III) min-TPC$(\emptyset, F, \infty)$ is strongly NP-hard for each fixed $|D| \geqslant 3$.

These results do not assume any structure on the conflict graph G_F. Motivated by an application, Vancroonenburg et al. (2014) assume that G_F is a complete bipartite graph; they prove that:

(I) both min-TPC$(\emptyset, F^B, \infty)$ and max-TPC$(\emptyset, F^B, \infty)$ remain NP-hard even if $s_i = 1$ for all $i \in S$ and $d_j = 3$ for all $j \in D$, and
(II) min-TPC$(\emptyset, F^B, \infty)$ remains NP-hard, even if edge set E is complete, all edge weights are equal and there are only 2 supply nodes with equal supply.

3.2.1 *Our Results*

Our results focus on the nature of the conflict graph G_F. First, we point out that, in the extreme cases where either F is empty (i.e. G_F contains no edges), or F contains all pairs (i.e. G_F is a clique), the resulting problem TPC$(\emptyset, F, u_{i,j})$ is easily seen to be polynomialy solvable.

Second, it is clear from our previous discussion that any graph class for which MIS is NP-hard, the corresponding problem TPC$(\emptyset, F, \infty)$ is NP-hard as well. However, there are graph classes for which independent set is easy, i.e. polynomialy solvable; thus, for these graph classes the

previous argument does not allow us to conclude that the corresponding special case of TPC is NP-hard. To deal with these graph classes, we provide the following results.

(I) TPC($\emptyset, F, \infty$) remains NP-hard, even if G_F is a matching (Theorem 3.1); this immediately implies NP-hardness of TPC($\emptyset, F, \infty$) in case G_F is a bipartite graph or an interval graph.
(II) TPC($\emptyset, F, \infty$) remains NP-hard, even if G_F is a simple path (Theorem 3.2); this immediately implies NP-hardness of TPC($\emptyset, F, \infty$) in case G_F is a tree.
(III) TPC($\emptyset, F, \infty$) remains NP-hard, even if G_F is a split graph (Theorem 3.3).

Notice that, indeed, MIS is polynomial solvable for the graph classes mentioned above. Further, in case that the conflict graph is complete bipartite, we close the gap between polynomial and NP-complete cases of TPC($\emptyset, F^B, \infty$):

(IV) TPC($\emptyset, F^B, \infty$) remains NP-hard, even if supply $s_i = 1$ for each supply node $i \in S$ and all demand $d_j = 2$ for each demand node $j \in D$, thereby settling a case left open by Vancroonenburg et al. (2014) (Theorem 3.4).

In addition, we give approximation results depending on the approximation factor for finding a certain independent set, for the following special cases of max-TPC:

(I) max-TPC($\emptyset, F_j, \infty$) with supply $s_i = 1$ for each $i \in S$ (Theorem 3.6);
(II) max-TPC($\emptyset, F_j, 1$) (Theorem 3.7);
(III) max-TPC(C, F_j, ∞) with supply $s_i = 1$ for each $i \in S$ and a valid q-colouring on the conflict graph induced by C (Theorem 3.8);
(IV) max-TPC($C, F_j, 1$) with a valid q-colouring on the conflict graph induced by C (Theorem 3.9).

3.3 COMPLEXITY RESULTS DEPENDING ON CONFLICT GRAPHS

As announced above, in this section we show that TPC($\emptyset, F, \infty$) is NP-hard in case G_F is a bipartite graph, interval graph, a matching, a simple path (Section 3.3.1) or a split graph (Section 3.3.2). In Section 3.3.3, we show that TPC($\emptyset, F^B, \infty$) remains NP-hard even if $s_i = 1$ for all $i \in S$ and $d_j = 2$ for all $j \in D$.

All proofs in this section assume a one-sided conflict set F (thus, conflict sets C_i are empty) and there is an infinite capacity on the edges. In fact, observe that to specify an instance of the resulting TPC problem it is sufficient to specify S (the set of supply nodes), D (the set of demand nodes), F (the conflict set), and w (the edge weights, edge set E is complete).

3.3.1 *When the Conflict Graph is a Matching or Simple Path*

In this section we show that even if the induced conflict graph G_F is a perfect matching or a simple path, both max-TPC and min-TPC remain NP-hard. This implies that the problem remains NP-hard for bipartite graphs, interval graphs and trees. We first prove that max-TPC and min-TPC remain NP-hard even if the induced conflict graph of F is a matching (only contains disconnected edges); after that we show that it is not difficult to extend the proof such that the results hold for a simple path as well.

To prove the theorem below we reduce our problem to monotone 1-in-3SAT, which is known to be NP-Complete (Garey and Johnson, 1979). An instance is defined as follows.

GIVEN: A set of n variables $\{v_1, \ldots, v_n\}$, and a set of m clauses $\{c_1, \ldots, c_m\}$, each containing 3 variables, each variable in positive form (unnegated).

QUESTION: Is there a truth-assignment that satisfies all clauses in C such that in each clause exactly 1 variable is TRUE?

Theorem 3.1. *Min-TPC($\emptyset, F, \infty$) is NP-hard, even if the induced conflict graph of G_F is a (perfect) matching.*

Proof. Given an instance I of monotone 1-in-3SAT, let λ_v be the number of clauses in which variable v occurs. We construct an instance of min-TPC($\emptyset, F, \infty$) as follows (see also Figure 11).

S: For each variable v, we have supply nodes $i(v)$, $i(\bar{v})$, each with supply λ_v. Hence we have $2n$ supply nodes.

D: For each clause $c \in C$ we have a demand node $j(c)$ with demand 1, and a demand node $j(\bar{c})$, with demand 2. And, for each variable v we have a demand node $j(v)$, with demand λ_v. Hence, we have $2m + n$ demand nodes.

F: The conflict set contains forbidden pairs $(i(v), i(\bar{v}))$, for every variable v. Note that the conflict graph G_F is indeed a perfect matching.

E: The weights on the edges are defined as follows:

- For each variable v that occurs in clause c we have $w_{i(v),j(c)} = 0$ and $w_{i(\bar{v}),j(\bar{c})} = 0$.
- For each variable v we have $w_{i(v),j(v)} = 0$ and $w_{i(\bar{v}),j(v)} = 0$.
- For any other edge $e \in E$ we have $w_e = 1$.

We now argue that an instance of monotone 1-in-3SAT is a YES-instance if and only if the corresponding instance of min-TPC($\emptyset, F, \infty$) has a solution with cost 0.

Suppose we have a YES-instance of monotone 1-in-3SAT, with a certificate indicating whether variable v is set to TRUE or FALSE. We create the following solution of the corresponding TPC instance.

If variable v is set to TRUE, then:

- For each clause c in which variable v occurs, 1 unit of supply is sent from supply node $i(v)$ to demand node $j(c)$,

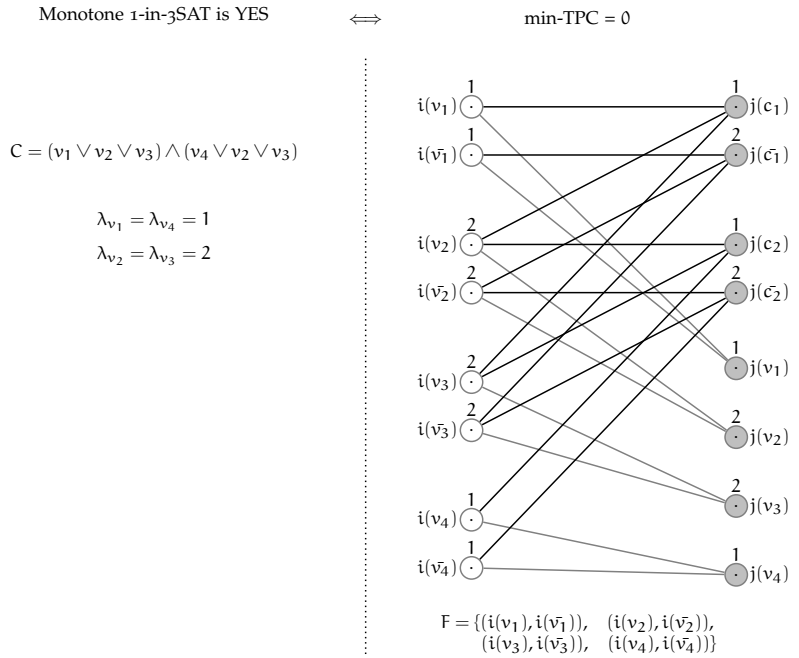


Figure 11: monotone 1-in-3SAT and TPC, only edges with weight 0 are shown

- λ_v units of supply are sent from supply node $i(\bar{v})$ to demand node $j(v)$.

If variable v is set to FALSE, then:

- For each clause c in which variable v occurs, 1 unit of supply is sent from supply node $i(\bar{v})$ to demand node $j(\bar{c})$,

- λ_v units of supply are sent from supply node $i(v)$ to demand node $j(v)$.

It is easy to see that no conflicts occur in this solution. Hence we only need to show that all supply and demand constraints are fulfilled.

Since λ_v equals the number of clauses containing variable v, we know that all supply constraints are fulfilled and that the demand for each demand node $j(v)$ is fulfilled.

We now show that the demand constraints are fulfilled as well. Since we have a YES-certificate, we know that exactly 1 variable is set to TRUE for each clause c. Hence, the demand for each demand node $j(c)$ is fulfilled. Furthermore if follows that in each clause c two variables are set FALSE, hence, the demand for each demand node $j(\bar{c})$ is fulfilled as well.

Observe that since we only use edges of weight 0, the cost of the solution is 0.

Now, suppose that we have a solution with cost 0 of the TPC instance.

This solution fulfills all supply and demand constraints and only uses edges of weight 0. Thus, we know that either λ_v units of supply are sent from $i(v)$ to $j(v)$, or from $i(\bar{v})$ to $j(v)$. (Since $i(v)$ and $i(\bar{v})$ are in conflict, there is no solution in which both $i(v)$ and $i(\bar{v})$ send supply to $j(v)$.) We will construct a truth-assignment as follows: for each variable v, if $i(v)$ sends supply to $j(v)$, then we set variable v to FALSE, else we set variable v to TRUE. We now argue that this is indeed a YES-certificate for monotone 1-in-3SAT.

It is easy to see that each clause has at least one variable set to TRUE. Indeed, the solution of the corresponding TPC instance with cost 0 is feasible, hence all demand constraints are fulfilled. Thus, demand node $j(c)$ receives one unit of supply from one of the supply nodes $i(v)$. Hence, for each clause at least one variable is set to TRUE.

Likewise, we know that all supply constraints are fulfilled. Thus, each supply node corresponding to a variable v that is set to TRUE distributes λ_v units of supply over demand nodes $j(c)$. Hence, the sum of λ_v of the variables v that are set to TRUE equals m, the number of clauses. Hence, each clause has exactly one variable set to TRUE. □

Corollary 3.1. *There is no constant-factor approximation algorithm for min-TPC($\emptyset, F, \infty$), even if the induced conflict graph of F is a (perfect) matching, unless P=NP.*

Corollary 3.2. *Max-TPC($\emptyset, F, \infty$) is NP-hard, even if the induced conflict graph G_F is a (perfect) matching.*

Proof. For max-TPC($\emptyset, F, \infty$) the reduction is basically the same, the only change are the edge weights. The weights on the edges are defined as follows:

- For each variable v that occurs in clause c we have $w_{i(v),j(c)} = 1$ and $w_{i(\bar{v}),j(\bar{c})} = 1$.
- For each variable v we have $w_{i(v),j(v)} = 1$ and $w_{i(\bar{v}),j(v)} = 1$.
- For any other edge $e \in E$ we have $w_e = 0$.

Note that now we have that a monotone 1-in-3SAT instance is a YES-instance, if and only if max-TPC($\emptyset, F, \infty$) has solution of profit $\sum_{v \in V} 2\lambda_v$. □

It follows immediately from Theorem 3.1 and Corollary 3.2 that TPC($\emptyset, F, \infty$) remains NP-hard even if the induced conflict graph G_F is a bipartite graph or an interval graph.

Let us now informally discuss the version of TPC($\emptyset, F, \infty$) for which the induced conflict graph of F is a simple path. Given an instance of TPC($\emptyset, F, \infty$), of which the induced conflict graph of F only contains disconnected edges, it is easy to create an instance of TPC($\emptyset, F, \infty$) of which the induced conflict graph F is a simple path. We can add $n - 1$ dummy supply nodes, each with supply 1, and one dummy demand node with demand $n - 1$ (recall that n is the number of variables in the monotone 1-in-3SAT instance). Edge set E is by definition a complete edge set, so we add the missing edges connecting the newly added dummy nodes. The edges connecting the dummy supply nodes to the dummy demand node will have weight 0, all other added edges will have weight 1. In the induced conflict graph we connect the dummy

supply nodes to the non-dummy supply nodes, such that the induced conflict graph becomes a simple path (see Figure 12). Observe that this does not affect the equivalence as described in the proof of the theorem above.

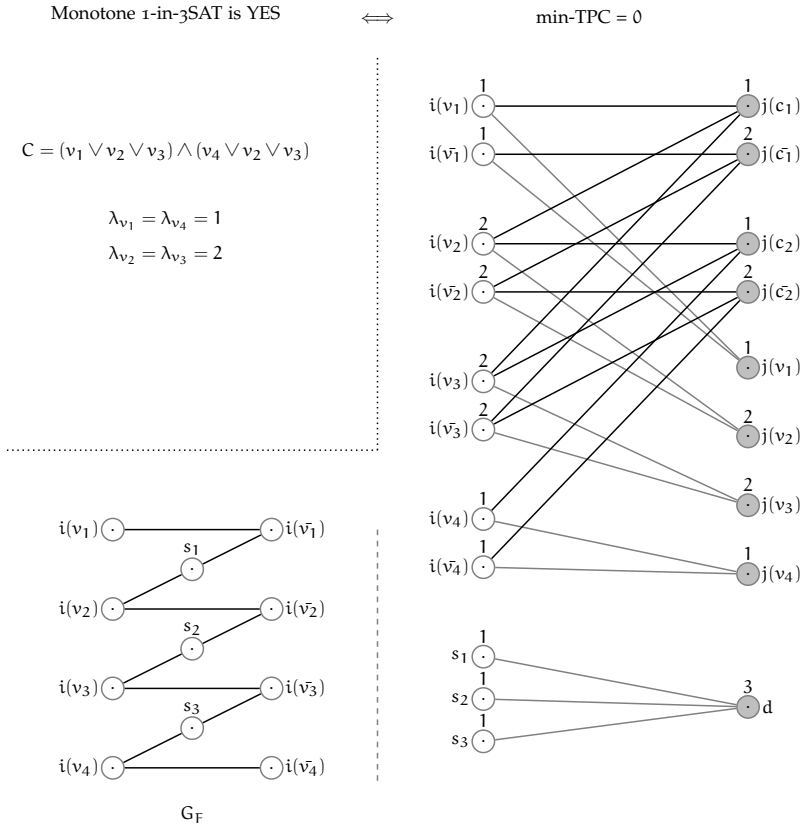


Figure 12: monotone 1-in-3SAT and TPC with simple path as induced conflict graph

Theorem 3.2. *TPC($\emptyset, F, \infty$) is NP-hard, even if the induced conflict graph of G_F is a simple path.*

Corollary 3.3. *There is no constant-factor approximation algorithm for min-TPC($\emptyset, F, \infty$), even if the induced conflict graph of F is a simple path (unless P=NP).*

It follows immediately from Theorem 3.2 and Corollary 3.3 that TPC($\emptyset, F, \infty$) remains NP-hard even if the induced conflict graph G_F is a tree.

3.3.2 *When the Conflict Graph is a Split Graph*

In this section we show that for every graph class $\mathcal{G}$ for which the so-called minimum sum colouring problem is NP-hard, the TPC restricted to conflict graphs from $\mathcal{G}$ is also NP-hard. For some graph classes (for instance bipartite graphs and interval graphs), this result provides an alternative proof of TPC($\emptyset, F, \infty$), since minimum sum colouring is NP-hard for these classes, see Bar-Noy and Kortsarz (1998); Marx (2005); Szkaliczki (1999). However, there exists a graph class not captured by Theorems 3.1 and 3.2 for which MIS is easy, while minimum sum colouring is NP-hard; the class of split graphs. We refer to Salavatipour (2000) where it is shown that minimum sum colouring is NP-hard for split graphs. Summarizing, the results in this section imply that TPC($\emptyset, F, \infty$) remains NP-hard even if the conflict graph is a split graph.

The minimum sum colouring problem is defined as follows.

GIVEN: A graph $G = (V, E_G)$ with $|V| = n$.
GOAL: Find a colouring $c : V \to \{1, \ldots, n\}$ of minimum weight $\sum_{v \in V} c(v)$ such that if $(v, u) \in E_G$, then $c(v) \neq c(u)$.

Theorem 3.3. *Let $\mathcal{G}$ be a graph class for which the minimum sum colouring problem is NP-hard. Min-TPC($\emptyset, F, \infty$) is NP-hard, even if the induced conflict graph $G_F \in \mathcal{G}$.*

Proof. Given an instance of the minimum sum colouring problem, specified by a graph $G = (V, E_G)$, we construct an instance of min-TPC$(\emptyset, F, \infty)$ as follows (see also Figure 13).

S: For each vertex $v \in V$ we have supply node $i(v)$ with supply 1. We also have an additional dummy node s with supply $n(n-1)$. Hence, we have $n+1$ supply nodes.

D: We have n demand nodes each with demand n (one for each colour). Hence, we have $D := \{1, \ldots, n\}$.

F: We have a forbidden pair $(i(v), i(u)) \in F$ for each edge in $(v, u) \in E_G$, i.e. $F := E_G$. Hence, the induced conflict graph G_F is equivalent to graph $G = (V, E_G)$.

E: The weights on the edges are defined as follows:

- for each $i(v) \in S \setminus \{s\}$ and for each $j \in D$: $w_{i(v),j} = j$;
- for each $j \in D$: $w_{s,j} = 0$.

The goal is to find a feasible solution of minimum cost. This specifies the instance of min-TPC.

We will now argue that a solution of minimum sum colouring with cost Z corresponds to a solution of min-TPC with cost Z and vice versa.

Suppose we have a feasible solution for the minimum sum colouring problem instance, meaning that there is a colour $c(v)$ assigned to each $v \in V$; the resulting cost of this solution is $Z = \sum_{v \in V} c(v)$. In the corresponding instance of min-TPC we send the single unit of supply of node $i(v) \in S$ to node $j \in D$ if vertex v is coloured with colour j, i.e. if $c(v) = j$; the weight of this flow equals j. Further, we distribute the supply from node s to all nodes $j \in D$ such that all supply and demand constraints are fulfilled; the weight of this flow equals 0. Since the colouring is a valid colouring in $G = (V, E)$, we know that vertices adjacent in G received distinct colours; in other words, no demand node receives supply from two supply nodes that are a forbidden pair in F. Also, the total value of this solution equals Z.

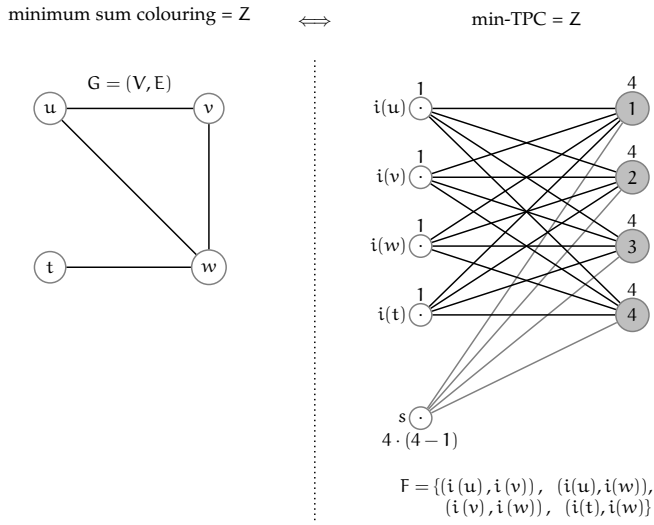


Figure 13: Example of construction, where $n = 4$

Now, suppose we have a feasible solution for our min-TPC instance with a total value Z. Then each $i(v) \in S$ sends a single unit of flow to exactly one demand node $j \in D$ (supply equals 1); the weight of this flow equals j. In the corresponding minimum sum colouring instance we give vertex $v \in V$ colour j if supply node $i(v) \in S$ sends supply to demand node $j \in D$. Since no conflicts are allowed in our assignment, we know that no two vertices adjacent in G receive the same colour, and the total cost of this solution, $\sum_v c(v)$, equals Z. □

Corollary 3.4. *Let $\mathcal{G}$ be a graph class for which the minimum sum colouring problem is NP-hard. Max-TPC($\emptyset$, F, ∞) is NP-hard, even if the induced conflict graph $G_F \in \mathcal{G}$.*

Proof. For max-TPC the reduction is basically the same, the only change are the edge weights. For max-TPC we define the weight on the edges as follows:

- for each $i(v) \in S \setminus \{s\}$ and for each $j \in D$: $w_{i(v),j} = n - j$.

- for each $j \in D$: $w_{s,j} = 0$.

Note that in this case the profit of the assignment in max-TPC does not equal the cost of the solution to the minimum colouring problem. However, it still holds that, if we are able to solve max-TPC in polynomial time, we will find an optimal solution to minimum sum colouring in polynomial time as well. □

It follows immediately from Theorem 3.3 and Corollary 3.4 that TPC($\emptyset, F, \infty$) remains NP-hard even if the induced conflict graph G_F is a split graph. Note that in Section 3.3.1 we have already shown that TPC($\emptyset, F, \infty$) remains NP-hard even if the induced conflict graph G_F is a bipartite or interval graph.

3.3.3 *The Conflict Graph is Complete Bipartite and* $s_i = 1$, $d_j = 2$

As mentioned before TPC($\emptyset, F^B, \infty$) is also known as the Red-Blue Transportation Problem (RBTP). Recall that for RBTP we know that the problem is NP-hard even if all supply $s_i = 1$ for all supply nodes $i \in S$ and all demand $d_j = 3$ for all demand nodes $j \in D$.

Vancroonenburg et al. (2014) state that it remains open whether the problem is still NP-hard even if all demand equals 2. We prove here that this problem remains NP-hard. To do so we make a reduction from a special case of SAT, known as (2,2)-SAT, which is proven to be NP-hard by Ratner and Warmuth (1990). An instance I of (2,2)-SAT is defined as follows.

GIVEN: A set of n variables $\{v_1, \ldots, v_n\}$ and a set of m clauses $C = \{c_1, \ldots, c_m\}$, such that every variable occurs exactly twice in positive form and exactly twice in negative form.
QUESTION: Is there a truth-assignment that satisfies all clauses in C?

Theorem 3.4. *Min-TPC($\emptyset, F^B, \infty$) is NP-hard, even if* $s_i = 1$ *and* $d_j = 2$ *for all* $i \in S$ *and* $j \in D$.

Proof. We stress that (2,2)-SAT does not impose any bound on the size of the clauses. Without loss of generality we assume that $|C|$ is even.

Recall that the induced conflict graph G_{F^B} is a complete bipartite graph. Hence, it is sufficient to define two sets of supply nodes, S_R and S_B. Then, the conflict set F^B contains a conflict for each pair of nodes with one node in S_R and one node in S_B. We shall split the set of supply nodes S into red supply nodes, S_R, and blue supply nodes, S_B.

Given instance I of (2,2)-SAT we construct an instance of min-TPC$(\emptyset, F^B, \infty)$, where all supply is 1 and all demand is 2, as follows.

S_R: For each clause $c \in C$ we have a supply node $r(c)$, and for each variable v occurring positively in clause c we have supply node $r(v, c)$.

S_B: For each clause $c \in C$ we have a supply node $b(c)$, and for each variable v occurring negatively in clause c we have supply node $b(\bar{v}, c)$.

D: For each clause $c \in C$ we have demand node $j(c)$, and for each variable v we have demand node $j(v)$, and we have n dummy nodes j_i, for $i = 1, \ldots, n$.

E: The weights on the edges are defined as follows:

- For each variable v that occurs in positive form in clause c we have $w_{r(v,c),j(v)} = 0$ and $w_{r(v,c),j(c)} = 0$.
- For each variable v that occurs in negative form in clause c we have $w_{b(\bar{v},c),j(v)} = 0$ and $w_{b(\bar{v},c),j(c)} = 0$.
- For every clause c we have $w_{r(c),j(c)} = 0$ and $w_{b(c),j(c)} = 0$.
- Any edge connecting a supply node to a dummy demand node j_i ($i = 1, \ldots, n$) has weight 0.
- For any other edge $e \in E$ we have $w_e = 1$.

Note that each of the sets S_R, S_B and D contain $2n + m$ nodes, which is even, because we assumed that the number of clauses, m, is even.

We now argue that the (2,2)-SAT instance is a YES-instance, if and only if min-TPC$(\emptyset, F^B, \infty)$ admits a solution of value 0 (see also Figure 14).

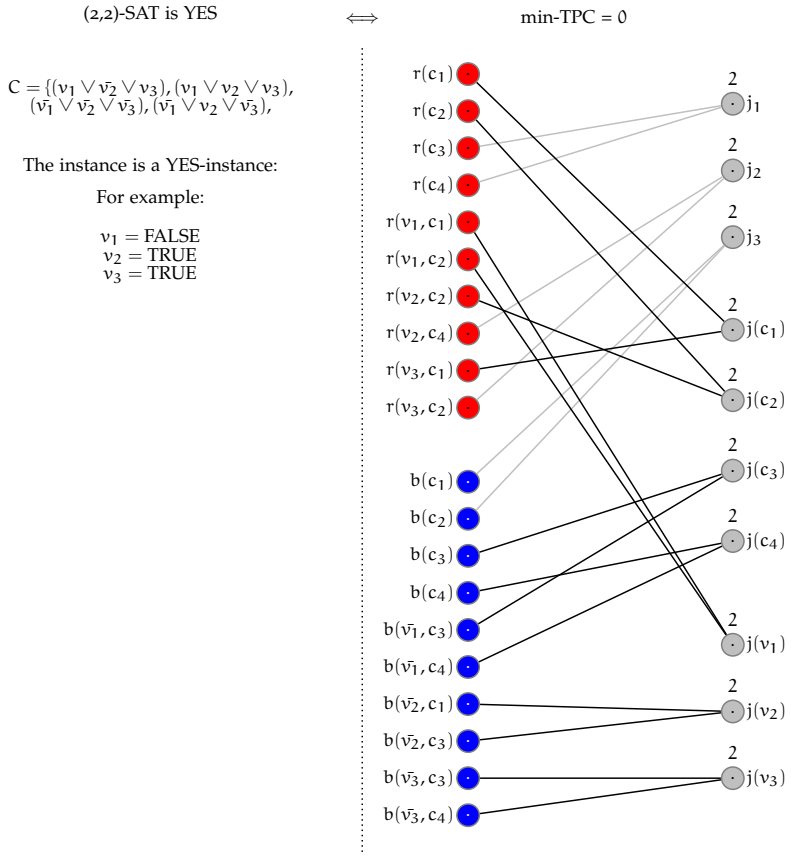


Figure 14: (2,2)-SAT and TPC$(\emptyset, F^B, \infty)$

First, suppose that we have a solution with value 0 of the corresponding TPC instance.

This solution fulfills all supply and demand constraints and only uses edges of weight 0. Hence, we know that a demand node $j(v)$ receives supply either from the two red supply nodes representing the two positive occurrences of variable v in a clause, or from the two blue supply nodes representing the two negative occurrences of variable v in a clause. If it receives supply from the two blue supply nodes we set variable v to be TRUE; otherwise we set variable v to be FALSE.

We now argue that this is a satisfying truth-assignment. Consider an arbitrary clause $c \in C$ and the corresponding demand node $j(c)$. Demand node $j(c)$ receives supply from at least one supply node representing the occurrence of a variable v in that clause. Suppose that supply node $r(v, c)$ sends supply to $j(c)$ and therefore no supply to $j(v)$. Hence variable v is set to TRUE and clause c in which v occurs in positive form is satisfied. Suppose $b(\bar{v}, c)$, sends supply to $j(c)$ and therefore no supply to $j(v)$. Hence variable v is set to FALSE and clause c in which it occurs in negative form is satisfied.

Second, suppose we have a satisfying truth-assignment.

If variable v has been set to TRUE, then demand node $j(v)$ receives supply from the two blue supply nodes representing the two negative occurrences of variable v. If variable v has been set to FALSE, then demand node $j(v)$ receives supply from the two red supply nodes representing the two positive occurrences of variable v.

Now consider a clause c and pick one variable v that satisfies this clause. If variable v occurs in positive form, then we know v has been set to TRUE in order to satisfy the clause. In this case both $r(v, c)$ and $r(c)$ send supply to demand node $j(c)$. If variable v occurs in negative form, then we know v has been set to FALSE in order to satisfy the clause. In this case both $b(\bar{v}, c)$ and $b(c)$ send supply to demand node $j(c)$.

So far we have assigned an even number of red supply nodes and an even number of blue supply nodes. The remaining supply nodes can

therefore be assigned to the remaining dummy demand nodes, such that each demand node receives supply of only one colour. □

Corollary 3.5. *There is no constant-factor approximation algorithm for min-TPC($\emptyset, F^B, \infty$), even if $s_i = 1$ and $d_j = 2$, unless P=NP.*

Corollary 3.6. *Max-TPC($\emptyset, F^B, \infty$) is NP-hard, even if $s_i = 1$ and $d_j = 2$.*

Proof. For max-TPC the reduction is basically the same, the only change is the edge weights. The weights on the edges are defined as follows:

- For each variable v that occurs in positive form in clause c we have $w_{r(v,c),j(v)} = 1$ and $w_{r(v,c),j(c)} = 1$.
- For each variable v that occurs in negative form in clause c we have $w_{b(\bar{v},c),j(v)} = 1$ and $w_{b(\bar{v},c),j(c)} = 1$.
- For every clause c we have $w_{r(c),j(c)} = 1$ and $w_{b(c),j(c)} = 1$.
- Any edge connecting a supply node to a dummy demand node has weight 1.
- For any other edge $e \in E$ we have $w_e = 0$.

Note that now we have that (2,2)-SAT instance is a YES-instance, if and only if max-TPC($\emptyset, F^B, \infty$) has solution of value $2(2n + m)$. □

3.4 APPROXIMATION FOR MAX-TPC

From the previous section we conclude that there exists no constant-factor approximation for min-TPC (unless P=NP). When it comes to max-TPC, we have shown in Section 3.2 that for any instance of the Maximum Independent Set Problem, there exists an equivalent instance of Max-TPC($\emptyset, F, \infty$). For the maximum independent set problem on graph $G = (V, E)$ with $|V| = n$, it is known that it is hard to approximate in polynomial time within a factor $n^{1-\epsilon}$, for any $\epsilon > 0$, unless P = NP (Håstad, 1996). Hence for max-TPC($\emptyset, F, \infty$) we know that no constant-factor approximation algorithm exists, even if $|D| = 1$, unless P=NP.

Theorem 3.5. *Max-TPC($\emptyset, F, \infty$), with $n := |S|$, is hard to approximate in polynomial time within a factor $n^{1-\epsilon}$, for any $\epsilon > 0$, even if supply $s_i = 1$ for each supply node $i \in S$, and $|D| = 1$ with $d_1 = n$ (unless P=NP).*

We now describe the so-called Separable Assignment Problem and show how existing approximation results can be applied to max-TPC.

3.4.1 *Max-TPC Meets Separable Assignment Problem*

The Separable Assignment Problem (SAP) generalizes several well-known assignment problems. An informal description of SAP is as follows. We are given a set of bins, a set of items and a profit for assigning a certain item to a certain bin. We are also given a packing constraint for each bin. Such a packing constraint must be such that every subset of a feasible packing is also feasible. So, for example, a packing constraint cannot state that each bin contains exactly 4 items, because a subset of 3 items would no longer be feasible. A packing constraint can state that each bin contains at most 4 items. The goal is to find an assignment of items to bins that maximizes total profit.

Suppose that, for a single bin, there is a β-approximation algorithm for finding the maximum profit assignment. Fleischer et al. (2011) present the following results for SAP:

(I) a $\beta(1 - \frac{1}{e})$-approximation algorithm (based on Randomized LP-Rounding), and
(II) a $(\frac{\beta}{\beta+1} - \epsilon)$-approximation algorithm (based on Local Search).

Thus, with $\gamma = \max\{\beta(1 - \frac{1}{e}), (\frac{\beta}{\beta+1} - \epsilon)\}$, we summarize the results of Fleischer et al. (2011) by saying that they provide a γ-approximation algorithm for SAP. Note that, if $\beta \geqslant \frac{1}{e-1}$ then $\gamma = \beta(1 - \frac{1}{e})$, else $\gamma = \frac{\beta}{\beta+1} - \epsilon$.

Let us now consider the following special case of max-TPC($\emptyset, F_j, \infty$), where all supply $s_i = 1$ for each supply node $i \in S$. We now show that we can describe this problem as a special case of SAP.

To do so, we view each supply node $i \in S$ in max-TPC as an item v_i in SAP and we view each demand node $j \in D$ with demand d_j in max-TPC as a bin b_j in SAP.

The packing constraint of each bin b_j states that;

(I) the total number of packed items does not exceed d_j, and
(II) a pair of items v_{i_1}, v_{i_2} can only be packed in bin b_j if the corresponding pair of supply nodes does not occur in the conflict set of the corresponding demand node, i.e. if $(i_1, i_2) \notin F_j$.

Note that, in our setting, finding the maximum profit solution for a single demand node $j \in D$ is equivalent to finding a maximum weight independent set of size at most d_j in conflict graph G_{F_j}, where the weight of a node $i \in S$ equals edge weight $w_{i,j}$. Finding a maximum weight independent set of size at most a given cardinality d (called MWIS-d) is an interesting combinatorial optimization problem in its own right; notice that this problem is more general than the well-studied maximum weight independent set problem (MWIS). Indeed, Kalra et al. (2017) study MWIS-d, and show that for bipartite graphs, MWIS-d is NP-complete (while MWIS is well-known to be solvable in polynomial time for bipartite graphs (Ahuja et al., 1993)). In addition, Kalra et al. (2017) give a $1/2$-approximation algorithm for MWIS-d on bipartite graphs. Such approximation results can be used to obtain constant factor approximation algorithm for a single bin, which in turn leads to approximation results for special cases of max-TPC using Fleischer et al. (2011).

Likewise, we can use the results of a generalization of SAP, called k-SAP. In k-SAP we are given a set of bins, a set of items I, a profit for assigning an item to a certain bin and a packing constraint for each bin. Now each item $i \in I$ may be assigned to at most k_i different bins, but at most once to each bin. The goal is to find an assignment of items to bins that maximizes total profit.

Again, assume that we are given a β-approximation algorithm for finding the maximum profit assignment for a single bin, and let $k =$

$\min_{i \in I} k_i$. Bender et al. (2015) prove that there exists a $\beta(1 - \frac{1}{e^k})$-approximation algorithm based on Randomized LP-Rounding.

One-sided max-TPC

Let us first consider only one-sided constraints. The above discussion will give us the following result.

Theorem 3.6. *Suppose there is a β_j-approximation for finding a maximum weight independent set of size at most d_j on the conflict graph G_{F_j} induced by F_j for each $j \in D$, let $\beta = \min_{j \in D} \beta_j$ and let $\gamma = \max\{\beta(1 - \frac{1}{e}), (\frac{\beta}{\beta+1} - \epsilon)\}$.*

Then there exists a γ-approximation algorithm for max-TPC($\emptyset, F_j, \infty$) with supply $s_i = 1$ for each supply node $i \in S$.

As mentioned before, suppose that in each of the conflict graphs G_{F_j} induced by F_j for each $j \in D$ we can find a maximum weight independent set of size at most d_j in polynomial time (i.e. $\beta = 1$ and $\gamma = (1 - \frac{1}{e})$), then we have a $(1 - \frac{1}{e})$-approximation algorithm. For example, this is the case when for all $j \in D$ we have that demand d_j is constant, i.e. not part of the input.

We can also apply the above result to max-TPC($\emptyset, F^B, \infty$), where the conflict graph is a complete bipartite graph $G = (R, B, E)$ (the Red-Blue Transportation problem). To find the most profitable assignment for a single demand node with demand d_j we compute the most profitable assignment with supply nodes in R, the most profitable assignment with supply nodes in B and choose the most profitable assignment of the two. Hence, we can find a maximum weight independent set of size at most d_j in polynomial time (i.e. $\beta = 1$ and $\gamma = (1 - \frac{1}{e})$). From the theorem above it follows we have an $(1 - \frac{1}{e})$-approximation algorithm, thereby improving the $1/2$ approximation factor found by Vancroonenburg et al. (2014).

Recall that one of the applications of the Red-Blue Transportation Problem is the assignment of male and female patients to hospital rooms, while keeping in mind that a room is either male-only or female-only. Suppose now, for example, that a male and female patient do not mind

sharing a room, because they are married. This gives rise to a TPC instance with a conflict graph which is bipartite, but not complete. As mentioned before, finding an Independent Set of size at most d_j of maximum weight in a bipartite graph is NP-hard and a greedy $1/2$-approximation exists (Kalra et al., 2017) (i.e. $\beta = 1/2$ and $\gamma = 1/3 - \epsilon$). Thus, Theorem 3.6 implies the existence of a $(1/3 - \epsilon)$ approximation algorithm for max-TPC($\emptyset, F_j, \infty$) with bipartite conflict graphs and $s_i = 1$ for all $i \in S$.

In case the supply nodes have a supply of $s_i \geqslant 1$ for all $i \in S$, and all edge capacities equal 1, we can use existing results for k-SAP to derive approximation results.

Theorem 3.7. *Suppose there is a β_j-approximation for finding a maximum weight independent set of size at most d_j on the conflict graph G_{F_j} induced by F_j for each $j \in D$. Let k be the minimum of all supply, i.e. $k = \min_{i \in S} s_i$, and let $\beta = \min_{j \in D} \beta_j$.*

Then there exists a $\beta(1 - \frac{1}{e^k})$-approximation algorithm for max-TPC($\emptyset, F_j, 1$).

Two-sided max-TPC

Recall that we require the conflicts to be one-sided in order to be able to use the results known for SAP. However, we can be slightly less restrictive. Assume that the demand nodes are coloured either red or blue and that each supply node may send flow to either only red demand nodes or only blue demand nodes. One approach is to solve the problem once while ignoring the blue demand nodes and once while ignoring the red demand nodes, and select the best of both solutions; this is a $1/2 \cdot \gamma$-approximation algorithm.

More generally, suppose we have two-sided conflicts, but one of the sides has identical conflicts, w.l.o.g. conflict set $C_i = C$ for all supply nodes $i \in S$. If we have a valid q-colouring on the induced conflict graph G_C, then we can split the problem into q one-sided max-TPC problems, which we can solve independently. A valid q-colouring on

a graph $G = (V, E)$, is a function $c : V \to \{1, \ldots, q\}$, such that for all $(v, u) \in E$ we have $c(v) \neq c(u)$.

Note that, to find a q-colouring, it need not be sufficient to know that a graph is q-colourable, since it is NP-hard to color a q-colourable graph with $q^{(\log q)/25}$ colours, for a sufficiently large constant q (Khot, 2001).

Theorem 3.8. *Suppose there is a β_j-approximation for finding a maximum weight independent set of size at most d_j on the conflict graph G_{F_j} induced by F_j for each $j \in D$, and a valid q-colouring on the conflict graph induced by C, let $\beta = \min_{j \in D} \beta_j$ and let $\gamma = \max\{\beta(1 - \frac{1}{e}), (\frac{\beta}{\beta+1} - \epsilon)\}$.*

Then there exists a $1/q \cdot \gamma$-approximation algorithm for max-TPC(C, F_j, ∞) with supply $s_i = 1$ for each supply node $i \in S$.

Again, we can strengthen these results if we allow supply nodes to have a supply $s_i \geqslant 1$ for $i \in S$, but restrict that each supply node is assigned at most once to each demand node, i.e. edge capacity $u_{i,j} = 1$. This allows us to use the results for k-SAP.

Theorem 3.9. *Suppose there is a β_j-approximation for finding a maximum weight independent set of size at most d_j on the conflict graph G_{F_j} induced by F_j for each $j \in D$, and a valid q-colouring on the conflict graph induced by C. Let k be the minimum of all supply, i.e. $k = \min_{i \in S} s_i$, and let $\beta = \min_{j \in D} \beta_j$.*

Then there exists a $1/q \cdot \beta(1 - \frac{1}{e^k})$-approximation algorithm for max-TPC$(C, F_j, 1)$.

3.5 CONCLUSION

In this chapter we introduce the Transportation Problem with Conflicts (TPC), which is a natural generalization of the classical Transportation problem. TPC is able to model several applications that have restrictions on the set of suppliers supplying a demand node and vice versa. Since in general TPC is at least as difficult as Maximum (Weighted) Independent Set, we are interested in establishing the complexity of TPC when the

restrictions satisfy particular structures that we capture using a conflict graph.

In particular, we have looked at the structure of the conflict graph induced by the conflict sets. We show that one-sided identical TPC remains NP-hard even if its conflict graph is a bipartite graph, planar graph, interval graph or a simple path.

Finally we use results known for the Separable Assignment Problem, and MWIS-d, to find approximation algorithms for special cases of max-TPC.

4 PARTITIONING VECTORS INTO QUADRUPLES

> I've always been fascinated with the idea that complexity can come out of such simplicity.
>
> *Will Wright*

4.1 INTRODUCTION

Partitioning Vectors into Quadruples (PQ) is the problem of partitioning $4n$ given nonnegative vectors $v_1, \ldots, v_{4n}$, each consisting of d components, into n clusters, each containing exactly four vectors. We refer to such a cluster of four vectors as a *quadruple* or a *quad* for short. The cost of a quad $Q = \{v_{i_1}, v_{i_2}, v_{i_3}, v_{i_4}\}$ is the sum of the component-wise maxima of the four vectors in the quad. The goal of our problem is to find a partition of the $4n$ vectors into n quads such that the total cost of all quads is minimum. For an example of an instance with a solution see Figure 15.

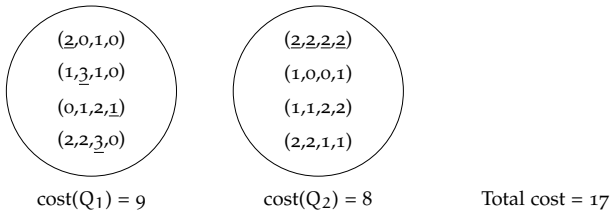


Figure 15: Partitioning Vectors into Quadruples, $n = 2$, $d = 4$

This chapter is the result of a collaboration with T. Erlebach, M. Mihalák and F. C. R. Spieksma.

We will analyze the following matching-based algorithm, called algorithm A, that finds a solution to problem PQ by proceeding in two phases.

In the first phase, algorithm A builds a complete, edge-weighted graph $G = (V, E)$ that has a node in V for each vector in the instance (hence $|V| = 4n$). The weight of an edge equals the sum of the component-wise maxima of the two vectors whose corresponding nodes span the edge. Now, algorithm A computes a minimum-cost perfect matching M^1 in the complete graph G, yielding $2n$ vector pairs. Let $p_1, \ldots, p_{2n}$ be the $2n$ matched vector pairs corresponding to the computed matching M^1.

In the second phase, algorithm A builds a complete, edge-weighted graph $G' = (V', E')$ that has a node in V' for each vector pair p_i found in the first phase ($i = 1, \ldots, 2n$; $|V'| = 2n$). The weight of an edge equals the sum of the component-wise maxima of the two vector pairs whose corresponding nodes span the edge. Now, algorithm A computes a minimum-cost perfect matching M^2 in the complete graph G', yielding n quadruples, which is a solution to the problem. Clearly, A is a polynomial-time algorithm, a rigorous description can be found in Section 4.2. It is not hard to see that algorithm A may fail to find an optimum solution for an instance of the problem, i.e., A is not exact. We are interested in analyzing how far off algorithm A's output can be from an optimum solution.

In this chapter we analyze the worst-case behavior of algorithm A for problem PQ. Our main result is a proof showing that A is a $3/2$-approximation algorithm, and that the analysis is tight. We also show that algorithm A has better approximation guarantees for various special cases of problem PQ. In particular, we show that A is $5/4$-approximation algorithm for the special case of PQ where each vector is a $\{0, 1\}$-vector containing exactly two ones, and the vectors, when seen as expressing the vertex/edge incidence matrix of a graph, correspond to edges of a simple, connected graph. We give a precise overview of our results in Section 4.2.4.

This chapter is organized as follows. The remainder of this section discusses related work that motivates our research, and Section 4.2 gives preliminaries, and states our results. In Section 4.3, we give the proofs of the upper bound on the worst-case ratio of algorithm A, while Section 4.4 contains the lower bound results. We conclude in Section 4.6.

4.1.1 *Related Literature*

Different problems in various fields are related to problem PQ, and share some of its characteristics. In addition, algorithm A can often be adjusted to work in a particular setting. We now review related literature and provide a number of such examples.

Onn and Schulman (2001) consider a problem where a given set of vectors in d-dimensional space, needs to be partitioned in a given number of clusters. The number of vectors in a cluster (its *size*) is not specified, and in addition, they assume that the objective function, which is to be maximized, is convex in the sum of the vectors in a same cluster. Their framework contains many different problems with diverse applications, and they show, for their setting, strongly polynomial-time exact algorithms. This is in contrast to our problem which is NP-hard (Section 4.2.2).

Another problem, distinct from, yet related to, our problem, comes from computational biology, and is described in Figueroa et al. (2005). Here, a component of a vector is a 0 or a 1 or an 'N'. In this setting neither the size of a cluster, nor the number of clusters is fixed; the goal is to find a partition of the set of vectors into a minimum number of clusters while satisfying the condition that a pair of vectors that is in the same cluster can only differ at a component where at least one of them has the value N. They prove hardness of this problem, and analyze the approximation behavior of heuristics for this problem.

Hochbaum and Levin (2010) describe a problem in the design of optical networks that is related to our special case where each vector is a $\{0,1\}$-vector containing two ones. In essence, their problem is to

cover the edges of a given bipartite graph by a minimum number of 4-cycles. They observe that this problem is a special case of unweighted 4-set cover; they give a $(13/10 + \epsilon)$-approximation algorithm (using local search), and analyze the performance of a greedy algorithm for a more general version of the problem. Our problem differs from theirs in the sense that we deal with a partitioning problem, where there is a weight for each set; in addition, our problem does not necessarily have a bipartite structure, nor do our quads need to correspond to 4-cycles.

Our problem is also intimately related to a problem occurring in wafer-to-wafer yield optimization (see, e.g., Reda et al. (2009) for a description). Central in this application is the production of so-called *waferstacks*, which is to be seen as a set of superimposed wafers. In our context, a wafer can be represented by a vector. A wafer consists of many dice (also called dies or die), each of which can be in two states: either functioning, i.e., good (which corresponds to a component in the vector with value '0'), or malfunctioning, i.e., bad (which corresponds to a a component in the vector with value '1'). The quality of a waferstack is measured by simply counting the number of components that have only 0's in the wafers contained in the waferstack. The goal is to partition the set of wafers into waferstacks (clusters) such that total quality is as high as possible. In this application, however, there are different types of wafers, and a waferstack needs to consist of one wafer of each type. This would correspond to an a priori given partition of the vectors. In addition, a typical waferstack consists in practice of many, i.e., more than 4, wafers. Dokka et al. (2014) analyze the worst-case behavior of different algorithms that have as a common feature solving assignment problems repeatedly. The case where there are three types of wafers, and the problem is to find waferstacks that are triples containing one wafer of each type is investigated in Dokka et al. (2013); for a particular objective function, they describe a $4/3$-approximation algorithm.

A special, yet very relevant special case of our problem is one where the edges of a given graph need to be partitioned into subsets each containing four edges (see Section 4.2 for a precise description). Indeed,

from a graph-theoretical perspective, there is quite some interest and literature in partitioning the edge set of a graph, i.e., to find an edge decomposition. In fact, edge decompositions where each cluster has prescribed size have already been studied in e.g. Jünger et al. (1985). Thomassen (2008) studies the existence of edge decompositions into paths of length 4, and Barát and Gerbner (2014) even study edge decompositions where each cluster is isomorphic to a tree consisting of 4 edges.

4.2 PRELIMINARIES

4.2.1 *Special Cases of Problem PQ*

We first observe that, for the analysis of algorithm A, we can restrict ourselves to instances of problem PQ where the $4n$ vectors are $\{0,1\}$-vectors. We call a vector non-negative when each of its entries is non-negative.

Lemma 4.1. *For any instance of problem PQ with arbitrary (rational) non-negative vectors there exists an equivalent instance with* $\{0,1\}$ *vectors of pseudo-polynomial length.*

Proof. If in a given instance the components of the input vectors are non-negative integers, we can create an equivalent instance with $\{0,1\}$-vectors of pseudo-polynomial length as follows. Consider any component k, $1 \leqslant k \leqslant d$, and let M_k be the largest value in the k-th component of any input vector. Then replace in each vector component k by M_k components, and a vector with value x in the k-th component is replaced by a vector that has x ones followed by $M_k - x$ zeros in the M_k components that replace component k. The resulting vectors are $\{0,1\}$-vectors, and the cost of any set of original vectors is the same as the cost of the corresponding set of modified vectors.

If the input vectors have non-negative rational values, we first multiply all vectors by the lowest common denominator of all the rational

numbers to make all vector components integers, and then use the construction described above. □

This shows that for a worst-case analysis of algorithm A, it is sufficient to consider $\{0,1\}$-vectors only. Indeed, any worst-case ratio of A shown to hold for $\{0,1\}$-vectors holds, using the argument of Lemma 4.1, for arbitrary rational nonnegative vectors. Of course, algorithm A can work directly with the original input vectors.

Thus, from hereon we restrict ourselves, without loss of generality, to the case of binary vectors. There are various special cases of PQ that are of independent interest. We will describe the particular special case in brackets following 'PQ'; we distinguish the following special cases.

- Problem PQ($\#1 \in \{1,2\}$). The case where each vector contains either one or two 1's; all other components have value 0. It will turn out that, at least in terms of the worst-case behavior of algorithm A, this special case displays the same behavior as problem PQ.
- Problem PQ($\#1 = 2$). The case where each binary vector contains exactly two 1's. Instances of this type can be represented by a multi-graph F with n nodes, each node corresponding to a component of a vector. Each vector is then represented by an edge spanning the two nodes that correspond to components with value 1. Of course, now a quad can be seen as a set of four edges, and its cost equals the number of nodes in the subgraph induced by these four edges.
- Problem PQ($\#1 = 2$, connected). The case where the graph F is connected.
- Problem PQ($\#1 = 2$, distinct). The case where the graph F is a simple graph. Equivalently, this means that each vector contains exactly two 1's and the vectors are pairwise distinct.
- Problem PQ($\#1 = 2$, distinct, connected). We distinguish a further special case by demanding that the graph F is also connected.

4.2.2 *The Complexity of Problem PQ*

Although our interest is on the worst-case behavior of algorithm A, it is relevant to establish the computational complexity of problem PQ. We prove hereunder that even its special case PQ(#1 = 2, distinct, connected) is NP-hard. This fact shows that no polynomial time algorithm for problem PQ can be exact, unless P=NP.

Theorem 4.1. *PQ(#1 = 2, distinct, connected) is NP-hard.*

Proof. As noted by Hochbaum and Levin (2010), the following problem was shown to be NP-complete by Holyer (1981). Given a connected, bipartite graph $G = (V, E)$, where $|E| = 4n$ for some $n \in \mathcal{N}$, does there exist a partition of the edge set E such that each set is isomorphic to a cycle on 4 nodes, i.e., a C_4? We refer to this decision problem as EPC_4.

Given an instance of EPC_4, we build the following instance of PQ(#1 = 2, distinct, connected). There are $4n$ vectors, each of length $|V|$. The $|V|$ components of each vector correspond to the nodes in V. Each edge in E gives rise to a vector whose entries are 0, except in the two components that correspond to the nodes spanning the edge; these components have value 1. This specifies all $4n$ vectors. The question is: does there exist a solution of this instance of PQ(#1 = 2, distinct, connected) with cost at most $4n$?

We claim that an instance of EPC_4 is a yes-instance if and only if there exists a solution to PQ(#1 = 2, distinct, connected) with cost at most $4n$. Indeed, if the instance of EPC_4 is a yes-instance, the four edges of each C_4 directly correspond to four vectors making up a quadruple with cost of 4, leading to a total cost of $4n$.

Consider now a solution to PQ(#1 = 2, distinct, connected), i.e., a set of n quads, with total cost $4k$. Since any four vectors are pairwise distinct, it follows that the four edges corresponding to each quadruple must span at least four vertices, i.e., each quadruple must have cost at least 4. And since total cost equals $4n$, it follows that each quadruple must have cost exactly 4. Finally, since the only possibility for four edges

to span four nodes in a simple bipartite graph is a C_4, it follows that a partition into C_4's must exist. □

4.2.3 *About Algorithm A: Notation and Properties*

Recall that, in our analysis, we may assume that all vectors are $\{0,1\}$-vectors. Let $v_i \vee v_j$ denote the vector that is the component-wise maximum of the two vectors v_i and v_j, i.e.:

$$v_i \vee v_j = (\max(v_{i,1}, v_{j,1}), \max(v_{i,2}, v_{j,2}), \ldots, \max(v_{i,d}, v_{j,d})).$$

Here, $v_{i,k}$ denotes the k-th component of vector v_i ($k = 1, \ldots, d$).

We use $|v_i|$ to denote the number of ones in vector v_i ($1 \leqslant i \leqslant 4n$), i.e.:

$$|v_i| = \sum_{k=1}^{d} v_{i,k}.$$

The cost of a quad $Q = \{v_1, v_2, v_3, v_4\}$ is then $\text{cost}(Q) = |v_1 \vee v_2 \vee v_3 \vee v_4|$. For a pair $p = \{v_1, v_2\}$ of vectors, we set $\text{cost}(p) = |v_1 \vee v_2|$.

For two vectors v_i and v_j, let $\text{sav}(v_i, v_j)$ (the "savings" made by combining v_i and v_j) denote the number of common ones in v and w, i.e.:

$$\text{sav}(v_i, v_j) = \sum_{k=1}^{d} \min(v_{i,k}, v_{j,k}).$$

If $p = \{v_1, v_2\}$ and $p' = \{v_3, v_4\}$ are pairs of vectors, we also write $\text{sav}(p, p')$ for $\text{sav}(v_1 \vee v_2, v_3 \vee v_4)$.

We prove the following observation concerning two $\{0,1\}$-vectors u and v explicitly:

Observation 4.1. $|u| + |v| = \text{sav}(u, v) + |u \vee v|$.

Proof. Let us partition the set of components that make up the vectors u and v into four sets:

- Those with a '1' in u, and a '0' in v: say there are $k_{u,\bar{v}}$ of them.

- Those with a '0' in u, and a '1' in v: say there are $k_{\bar{u},v}$ of them.
- Those with a '1' in u, and a '1' in v: say there are $k_{u,v}$ of them.
- Those with a '0' in u, and a '0' in v: say there are $k_{\bar{u},\bar{v}}$ of them.

Obviously, since $|u| = k_{u,\bar{v}} + k_{u,v}$, $|v| = k_{\bar{u},v} + k_{u,v}$, $|\mathrm{sav}(u,v)| = k_{u,v}$, and $|u \vee v| = k_{u,\bar{v}} + k_{\bar{u},v} + k_{u,v}$, the claim follows. □

Let us revisit the description of Algorithm A. In the first phase, it computes a minimum-cost perfect matching M^1 in the complete graph on the given $4n$ vectors, where the weight of the edge between vectors v_i and v_j is set to $|v_i \vee v_j|$. Let $p_1, \ldots, p_{2n}$ be the $2n$ matched vector pairs corresponding to the computed matching M^1, and let $\mathrm{cost}(M^1)$ denote the cost of the matching M^1. For $1 \leqslant i \leqslant 2n$, let v_i^1 and v_i^2 be the two vectors in the vector pair p_i, and let $v_i' = v_i^1 \vee v_i^2$.

In the second phase, Algorithm A computes a minimum-cost perfect matching M^2 in the complete graph on the $2n$ vector pairs, where the weight of the edge between pairs p_i and p_j is set to $|v_i' \vee v_j'|$. The quads corresponding to M^2 are output as a solution. Let $\mathrm{cost}(M^2)$ be the cost of matching M^2.

Observation 4.2. $A(I) = \mathrm{cost}(M^2)$ *and* $\mathrm{cost}(M^2) \leqslant \mathrm{cost}(M^1)$.

Lemma 4.2. *In the second phase of algorithm* A*, we can set the cost of the edge between* p_i *and* p_j *to be* $-\mathrm{sav}(v_i', v_j')$.

Proof. It follows from Observation 4.1 that the cost of any perfect matching M^2 is

$$\begin{aligned}
\mathrm{cost}(M^2) &= \sum_{(v_i', v_j') \in M^2} |v_i' \vee v_j'| \\
&= \sum_{(v_i', v_j') \in M^2} (|v_i'| + |v_j'| - \mathrm{sav}(v_i', v_j')) \\
&= \sum_i |v_i'| - \sum_{(v_i', v_j') \in M^2} \mathrm{sav}(v_i', v_j') \\
&= \mathrm{cost}(M^1) - \sum_{(v_i', v_j') \in M^2} \mathrm{sav}(v_i', v_j').
\end{aligned}$$

Therefore, finding a matching that minimizes $\sum_{(v'_i,v'_j)\in M^2} |v'_i \vee v'_j|$ is equivalent to finding a matching that maximizes $\sum_{(v'_i,v'_j)\in M^2} \mathrm{sav}(v'_i, v'_j)$. □

Let $\mathrm{savings}(M^2)$ denote the total savings of the perfect matching M^2, i.e.,

$$\mathrm{savings}(M^2) = \sum_{v'_i v'_j \in M^2} \mathrm{sav}(v'_i, v'_j).$$

Then, it follows from Lemma 4.2:

$$\mathrm{cost}(M^2) = \mathrm{cost}(M^1) - \mathrm{savings}(M^2). \tag{3}$$

Observation 4.2 and Equation (3) imply:

Corollary 4.1. $A(I) = \mathrm{cost}(M^1) - \mathrm{savings}(M^2)$.

In view of this corollary, it follows that if we can show that $\mathrm{cost}(M^1) \leqslant B$ and $\mathrm{savings}(M^2) \geqslant S$ for some bounds B and S, we can conclude that $A(I) \leqslant B - S$.

Two vectors u and v are *identical* when $u = v$, and a pair of identical vectors is called an *identical pair*. In the following we show that among the set of minimum-cost perfect matchings, there is one that contains a maximum number of identical pairs.

Lemma 4.3. *There is a minimum-cost perfect matching in* G *that contains a maximum number of identical pairs.*

Proof. Assume that, in a minimum-cost perfect matching, there are two identical vectors u and v that are not matched to each other; instead, let u be matched to some vector a, and v be matched to some vector b. We need to prove that, when $u = v$, the cost of the pairs $\{u, a\}$ and $\{v, b\}$ is at least as large as the cost of the pairs $\{u, v\}$ and $\{a, b\}$.

$$\begin{gathered} |u \vee a| + |v \vee b| = \\ |\mathrm{sav}((u \vee a), (v \vee b))| + |(u \vee a) \vee (v \vee b)| \geqslant \\ |u| + |u \vee a \vee b| \geqslant |u| + |a \vee b|. \end{gathered}$$

The first equality holds by Observation 4.1, the first inequality follows from $u = v$, and the second inequality is trivial. □

Thus, in the implementation of our algorithm A, we can first greedily match pairs of identical vectors as long as they exist, and then use any standard minimum-cost perfect matching algorithm to compute a perfect matching of the remaining vectors.

4.2.4 *Our Results*

In this chapter, we prove the following results.

Theorem 4.2. *Algorithm A is a $3/2$-approximation algorithm for problem PQ, and this bound is tight.*

Theorem 4.3. *Algorithm A is a $3/2$-approximation algorithm for problem $PQ(\#1 \in \{1, 2\})$, and this bound is tight.*

Theorem 4.4. *Algorithm A is a $4/3$-approximation algorithm for problem $PQ(\#1 = 2)$, and this bound is tight.*

Theorem 4.5. *Algorithm A is a $4/3$-approximation algorithm for problem $PQ(\#1 = 2, \textit{connected})$, and this bound is tight.*

Theorem 4.6. *Algorithm A is a $13/10$-approximation algorithm for problem $PQ(\#1 = 2, \textit{distinct})$, and its worst-case ratio is at least $5/4$.*

Theorem 4.7. *Algorithm A is a $5/4$-approximation algorithm for problem $PQ(\#1 = 2, \textit{distinct, connected})$, and this bound is tight.*

We summarize these statements in Table 9.

Proofs of the theorems are found in the next sections; the upper-bound proofs (Lemma's 4.4-4.7) are in Section 4.3, the lowerbound results (Observations 4.4-4.6) are in Section 4.4.

As an aside, we show in Section 4.5 that the worst-case ratio of a natural greedy algorithm, both for problem PQ, as well as for its special case PQ(#1 = 2, distinct, connected) is worse than the worst-case ratio of algorithm A.

Problem name	Lower Bound	Upper Bound
PQ	$3/2$	$3/2$ (Lem. 4.4)
PQ(#1 $\in \{1, 2\}$)	$3/2$ (Obs. 4.4)	$3/2$
PQ(#1 $= 2$)	$4/3$	$4/3$ (Lem. 4.5)
PQ(#1 $= 2$, connected)	$4/3$ (Obs. 4.5)	$4/3$
PQ(#1 $= 2$, distinct)	$5/4$	$13/10$ (Lem. 4.6)
PQ(#1 $= 2$, distinct, connected)	$5/4$ (Obs. 4.6)	$5/4$ (Lem. 4.7)

Table 9: Overview of bounds for algorithm A

4.3 UPPER BOUND APPROXIMATION GUARANTEE

In this section, we prove the upper bounds for the worst-case ratio's of algorithm A for problem PQ and its special cases. In Section 4.3.1 we prove the $3/2$ upper bound for the worst-case ratio of problem PQ, in Section 4.3.2 we prove the $4/3$ upper bound for the worst-case ratio of Problem PQ(#1 $= 2$), and in Section 4.3.3 we prove the $13/10$ upper bound for the worst-case ratio of Problem PQ(#1 $= 2$, distinct). Finally, in Section 4.3.4 we prove the $5/4$ upper bound for the worst-case ratio of Problem PQ(#1 $= 2$, distinct, connected).

4.3.1 *Analysis for PQ*

Lemma 4.4. *The worst-case ratio of algorithm A for PQ is at most $3/2$.*

Proof. We use the terminology from Section 4.2, where M^1 refers to the minimum-cost perfect matching found by A in the first phase based on the costs $|v_i \vee v_j|$, and M^2 refers to the maximum-weight perfect matching found in the second phase based on the savings $\text{sav}(v'_i, v'_j)$. As described in Corollary 4.1, we can express the cost of the solution found by algorithm A as follows:

$$A(I) = \text{cost}(M^1) - \text{savings}(M^2).$$

In this proof we will identify an alternative matching of vectors for the first phase, M^{A1}. We will also identify an alternative matching for the second phase, M^{A2}, meaning that M^{A2} is a matching of the vector pairs found in the first phase by the algorithm. Since M^1 is a minimum-cost matching and M^2 a maximum-weight matching we know that:

$$A(I) \leqslant \text{cost}(M^{A1}) - \text{savings}(M^{A2}). \tag{4}$$

Consider the quads in an optimum solution.

We obtain matching M^{A1} by specifying two vector pairs in each quad from the optimum solution. Obtaining matching M^{A2} is more involved. First, we will identify potential matches between vector pairs in M^1. These potential matches are represented as edges in an auxiliary graph H whose vertex set V' is the set of vector pairs resulting from the algorithm's matching of the first phase. Thus we will construct a graph $H = (V', E_1 \cup E_2)$ where the edge sets E_1 and E_2 will be described in detail. The weight of each edge e in the graph H, called $w(e)$, represents the savings that algorithm A realizes in the second phase if it were to match the vector pairs that are the endpoints of e. Any matching in this graph is an alternative matching for the second phase. We will select the matching with the highest weight to be M^{A2}.

The graph H will be bipartite and have maximum degree 2, implying that every cycle in H must have even length. We use E_1 to denote the edges in H whose both endpoints have degree one, with S_1 their total weight, i.e., $S_1 = \sum_{e \in E_1} w(e)$. We use E_2 to denote the remaining edges in H, with S_2 representing their total weight, i.e., $S_2 = \sum_{e \in E_2} w(e)$. Our construction will ensure that E_2 is a collection even-length cycles.

Claim 1. *Consider an undirected, edge-weighted graph H that is bipartite, and has maximum degree 2. Let S_1 (S_2) be the total weight of edges between nodes with degree 1 (degree 2), and let M^2 be a maximum-weight matching in H. Then,*

$$\text{savings}(M^2) \geqslant \text{savings}(M^{A2}) \geqslant S_1 + 1/2 \cdot S_2. \tag{5}$$

Proof. The claim follows because there exists a matching in H with savings $S_1 + 1/2 \cdot S_2$. This matching can be obtained by taking all edges from E_1 and partitioning E_2 into two matchings, taking the one with maximum savings. □

Inequalities (4) and (5) imply:

$$\begin{aligned} A(I) &= \text{cost}(M^1) - \text{savings}(M^2) \leqslant \text{cost}(M^{A1}) - \text{savings}(M^{A2}) \\ &\leqslant \text{cost}(M^{A1}) - (S_1 + 1/2 \cdot S_2). \end{aligned} \tag{6}$$

Consider the quantity $\text{cost}(M^{A1}) - (S_1 + 1/2 \cdot S_2)$, which - according to (6) - serves as an upper bound for the cost of the solution found by algorithm A. Informally speaking, we are going to distribute this quantity over the quads from the optimum solution: for each quad Q in the optimum solution, we will define its corresponding "share" of $\text{cost}(M^{A1}) - (S_1 + 1/2 \cdot S_2)$ by ϕ_Q; we will refer to ϕ_Q as the *contribution reserved for* Q. This contribution ϕ_Q consists of terms reflecting the contribution to M^{A1}, and terms reflecting the contribution to the total savings $S_1 + 1/2 \cdot S_2$. We will show that the choice of ϕ_Q satisfies, for each Q from the optimum solution:

$$\phi_Q \leqslant 3/2 \cdot \text{cost}(Q). \tag{7}$$

This leads to the following:

$$\begin{aligned} A(I) &\leqslant \text{cost}(M^{A1}) - (S_1 + 1/2 S_2) = \sum_Q \phi_Q \\ &\leqslant \sum_Q 3/2 \cdot \text{cost}(Q) = 3/2 \cdot \text{OPT}(I). \end{aligned}$$

The remainder of the proof is devoted to proving the above relationship. Thus, correctness hinges upon proving that

(i) the graph H is bipartite, and has maximum degree 2,

and that our choice of ϕ_Q satisfies

(ii) $\phi_Q \leqslant 3/2 \cdot \text{cost}(Q)$ for each Q from the optimum solution, and

(iii) $\sum_Q \phi_Q = \text{cost}(M^{A1}) - (S_1 + 1/2 \cdot S_2)$.

We now prove (i), (ii), and (iii).

Proving that the graph H is bipartite, and has maximum degree 2

Definition 4.1. *A **lucky pair** is a pair of vectors that are in the same quad in the optimum solution and that are matched by algorithm A in the first phase.*

Let us first construct the edge set E_1 of the graph H. Consider an optimal quad Q that contains two lucky pairs p_1 and p_2. By definition, p_1 and p_2 correspond to nodes in H, and we add the edge $e = (p_1, p_2)$ to E_1, and we set its weight equal to the corresponding savings: $w(e) = \text{sav}(p_1, p_2)$. Notice that neither node p_1 nor p_2 will be incident to any other edge in H.

Let us now proceed with the edge set E_2 of the graph H. The edge set E_2 of H will be constructed as follows. Consider an auxiliary multigraph K whose vertices are the quads of the optimal solution that contain at most one lucky pair. For every vector pair (v_1, v_2) that is matched by the algorithm in the first phase, add an edge (Q_1, Q_2) to K, where Q_i is the optimal quad that contains the vector v_i for $i = 1, 2$. We say that this edge (Q_1, Q_2) *corresponds* to the pair (v_1, v_2). If (v_1, v_2) is a lucky pair, the edge added to K is a self-loop at the node corresponding to the quad that contains v_1 and v_2. Every edge of K corresponds to a vertex of the auxiliary graph H, as the vertices of H are the pairs of vectors matched by the algorithm in the first phase. Note that each vertex in K has degree four, where a self-loop contributes 2 to the degree of the vertex to which it is attached. As every vertex of K has even degree, every connected component of K admits an Eulerian cycle. Note that each Eulerian cycle of a connected component of K has an even number of edges as each vertex in the component has degree 4 and the number

of edges in a multi-graph with self-loops is equal to half the sum of the vertex degrees.

Pick an arbitrary Eulerian cycle (possibly including self-loops) in each component of K. We will use these Eulerian cycles to determine edges to be added to E_2 in H in such a way that H is bipartite and has maximum degree 2. Orient each Eulerian cycle in an arbitrary way into a directed cycle. For every pair of consecutive edges (Q_i, Q_{i+1}) and (Q_{i+1}, Q_{i+2}) on such a cycle, where (Q_i, Q_{i+1}) corresponds to p_1 and (Q_{i+1}, Q_{i+2}) to p_2, add the edge $e = (p_1, p_2)$ to H, and set its weight to $w(e) = \mathrm{sav}(p_1, p_2)$. The edge (p_1, p_2) is considered to be added for the optimal quad Q_{i+1}, and its savings are used only in the analysis of that optimal quad. See Figure 16 for an example of graph K and H.

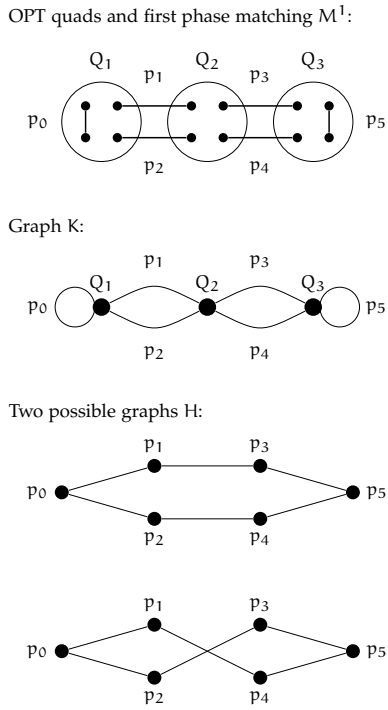


Figure 16: Example of graphs K and H with p_0 and p_5 as lucky pairs

Claim 2. *The graph* H *is bipartite, and has maximum degree 2.*

Proof. Observe that, for every component of K, the edges added to H form a single cycle consisting of an even number of edges. This is true because the graph of the edges added to H is isomorphic to the line graph of the Eulerian cycle and, as argued in the previous discussion, that cycle contains an even number of edges. □

Clearly, when building H as described above, different choices can be made, since there might be different Eulerian cycles possible in K in the case of a quad not containing a lucky pair. We need to be explicit about these different possibilities. Indeed, consider any quad $Q = \{v_1, v_2, v_3, v_4\}$ that does not contain a lucky pair. Let p_i be the vector pair containing v_i that was matched by the algorithm in the first phase, for $1 \leqslant i \leqslant 4$. Let (p_1, p_3) and (p_2, p_4) be the edges added to H, and observe that they lie on a single even-length cycle C in H (namely, the cycle created in H from the Eulerian cycle of the component of K that contains Q). Assume, without loss of generality, that after removing the edges (p_1, p_3) and (p_2, p_4) from H, the cycle C splits into two paths in H, one between node p_1 and node p_2, and one between node p_3 and node p_4. Define $\{\{p_1, p_2\}, \{p_3, p_4\}\}$ to be the *good partition* of Q.

Observe now that replacing the edges (p_1, p_3) and (p_2, p_4) in H by the edges (p_1, p_4) and (p_2, p_3) in H replaces the cycle C by another cycle C' consisting of the same number of edges. This means that any one of the two possible combinations of two independent edges between a vertex on one side of the good partition and a vertex on the other side of the good partition can be chosen for inclusion in E_2, while maintaining the property that E_2 consists of even-length cycles. For example, in Figure 16, we show the two possible cycles of even length for graph H.

Observe that the discussion following Claim 2 has identified a collection of graphs H, each satisfying the conditions of Claim 2.

Proving that $\phi_Q \leqslant 3/2 \cdot \text{cost}(Q)$ *for each quad* Q

Now we are ready to analyse the contribution to ϕ of each quad Q from the optimum solution. There are three types of quads in an optimum solution:

- those quads that contain two lucky pairs; let us refer to this set of quads as O_2,
- those quads that contain one lucky pair; let us refer to this set of quads as O_1,
- those quads that contain no lucky pairs; let us refer to this set of quads as O_0.

Let us first consider the quads from the set O_2. Let $Q \in O_2$ equal $\{p_1, p_2\}$. We set $p_1, p_2 \in M^{A1}$ and, by definition, E_1 contains (p_1, p_2), thus

$$\phi_Q = \text{cost}(p_1) + \text{cost}(p_2) - \text{sav}(p_1, p_2). \tag{8}$$

Since $\text{cost}(Q) = \text{cost}(p_1) + \text{cost}(p_2) - \text{sav}(p_1, p_2) = \phi_Q$, it trivially follows that:

$$\phi_Q \leqslant 3/2 \cdot \text{cost}(Q) \text{ for each quad } Q \in O_2. \tag{9}$$

Next, we consider the quads containing a single lucky pair, i.e., the quads from O_1. Thus, with $Q = \{u_1, u_2, v_1, v_2\}$, let the lucky pair be $p = \{u_1, u_2\}$, and let $p_1 = \{v_1, w_1\}$ and $p_2 = \{v_2, w_2\}$ be the other nodes in H that contain the vectors from quad Q. We select p and $\{u_1, u_2\}$ to be in matching M^{A1}. Clearly, the edge set E_2 contains the edge (p, p_1) as well as (p, p_2), with weights respectively $\text{sav}(p, p_1)$ and $\text{sav}(p, p_2)$. Observe that there are no other edges in H incident to node p.

Hence, for each $Q = \{u_1, u_2, v_1, v_2\} \in O_1$, we set:

$$\phi_Q = \text{cost}(p) + |v_1 \vee v_2| - 1/2(\text{sav}(p, p_1) + \text{sav}(p, p_2)). \tag{10}$$

The cost of Q is:

$$\text{cost}(Q) = \text{cost}(p) + |v_1 \vee v_2| - \text{sav}(p, v_1 \vee v_2). \tag{11}$$

Notice that $\text{cost}(Q) \geqslant \text{cost}(p)$ and $\text{cost}(Q) \geqslant |v_1 \vee v_2|$, and therefore:

$$\text{cost}(Q) \geqslant 1/2(\text{cost}(p) + |v_1 \vee v_2|). \tag{12}$$

Further, we have:

$$\text{sav}(p, v_1 \vee v_2) \leqslant \text{sav}(p, v_1) + \text{sav}(p, v_2) \leqslant \text{sav}(p, p_1) + \text{sav}(p, p_2). \tag{13}$$

Combining (10), (12), and (13) gives, for each quad $Q \in O_1$:

$$\begin{aligned}
\phi_Q &= \text{cost}(p) + |v_1 \vee v_2| - 1/2(\text{sav}(p, p_1) + \text{sav}(p, p_2)) \\
&\leqslant \text{cost}(p) + |v_1 \vee v_2| - 1/2\text{sav}(p, v_1 \vee v_2) \\
&= 1/2(\text{cost}(p) + |v_1 \vee v_2|) + 1/2(\text{cost}(p) + |v_1 \vee v_2| - \text{sav}(p, v_1 \vee v_2)) \\
&\leqslant \text{cost}(Q) + 1/2 \cdot \text{cost}(Q) = 3/2 \cdot \text{cost}(Q).
\end{aligned} \tag{14}$$

Now, consider a quad $Q = \{v_1, v_2, v_3, v_4\}$ from O_0, i.e., a quad with no lucky pairs. Hence we have the following three choices for the alternative first phase matching: $\{v_1, v_2\}, \{v_3, v_4\} \in M^{A1}$, $\{v_1, v_3\}, \{v_2, v_4\} \in M^{A1}$ or $\{v_1, v_4\}, \{v_2, v_3\} \in M^{A1}$.
Let p_i be the vector pair matched by the algorithm in the first phase that contains v_i, for $1 \leqslant i \leqslant 4$. Without loss of generality, assume that $\{\{p_1, p_2\}, \{p_3, p_4\}\}$ is the good partition associated with Q; it follows that we can choose for inclusion in H either the edges (p_1, p_3) and (p_2, p_4) or the edges (p_1, p_4) and (p_2, p_3). Depending on which pair of edges we choose for inclusion in H, the term in ϕ_Q that reflects the contribution to the savings equals either $1/2(\text{sav}(p_1, p_3) + \text{sav}(p_2, p_4))$ or $1/2(\text{sav}(p_1, p_4) + \text{sav}(p_2, p_3))$.

Observe that we have:

$$\mathrm{sav}(p_1, p_3) + \mathrm{sav}(p_2, p_4) \geqslant \mathrm{sav}(v_1, v_3) + \mathrm{sav}(v_2, v_4) \tag{15}$$

$$\mathrm{sav}(p_1, p_4) + \mathrm{sav}(p_2, p_3) \geqslant \mathrm{sav}(v_1, v_4) + \mathrm{sav}(v_2, v_3). \tag{16}$$

Summarizing, depending on which pairs of vectors from Q we put in M^{A1} (there are three possibilities), and which two edges we add to H (there are two possibilities), we can get (using (15)-(16)) any of the following six bounds on the contribution ϕ_Q of Q:

$$\phi_Q \leqslant |v_1 \vee v_2| + |v_3 \vee v_4| - 1/2(\mathrm{sav}(v_1, v_3) + \mathrm{sav}(v_2, v_4)), \tag{17}$$

$$\phi_Q \leqslant |v_1 \vee v_2| + |v_3 \vee v_4| - 1/2(\mathrm{sav}(v_1, v_4) + \mathrm{sav}(v_2, v_3)), \tag{18}$$

$$\phi_Q \leqslant |v_1 \vee v_3| + |v_2 \vee v_4| - 1/2(\mathrm{sav}(v_1, v_3) + \mathrm{sav}(v_2, v_4)), \tag{19}$$

$$\phi_Q \leqslant |v_1 \vee v_3| + |v_2 \vee v_4| - 1/2(\mathrm{sav}(v_1, v_4) + \mathrm{sav}(v_2, v_3)), \tag{20}$$

$$\phi_Q \leqslant |v_1 \vee v_4| + |v_2 \vee v_3| - 1/2(\mathrm{sav}(v_1, v_3) + \mathrm{sav}(v_2, v_4)), \tag{21}$$

$$\phi_Q \leqslant |v_1 \vee v_4| + |v_2 \vee v_3| - 1/2(\mathrm{sav}(v_1, v_4) + \mathrm{sav}(v_2, v_3)). \tag{22}$$

We will choose the pairs of vectors for M^{A1} and the two edges we add to E_2 for Q such that the smallest of these six bounds becomes ϕ_Q. For that choice, all right-hand sides (17)-(22) are upper bounds on the contribution ϕ_Q.

Since we may assume that each vector v_i is a $\{0, 1\}$-vector, and since the right-hand sides of (17)-(22) involve the four vectors of quad Q, there are $2^4 = 16$ possible configurations for the 4 values of a particular component in the four vectors. Thus, for each $j = 0, 1, \ldots, 15$, we can write its binary expansion as $j = 8b_{j,1} + 4b_{j,2} + 2b_{j,3} + b_{j,4}$. We denote by k_j the number of components of the vectors of the quad whose values equal the binary expansion of j, i.e., the number of components k with $v_{1,k} = b_{j,1}$, $v_{2,k} = b_{j,2}$, $v_{3,k} = b_{j,3}$ and $v_{4,k} = b_{j,4}$. For an example see Figure 17.

$$
\begin{array}{llllll}
\nu_1=(& \boxed{1,} & \boxed{0,} & \boxed{1,} & \boxed{0} &) \\
\nu_2=(& 1, & 1, & 1, & 0 &) \\
\nu_3=(& 0, & 1, & 0, & 0 &) \\
\nu_4=(& 0, & 1, & 0, & 1 &) \\
& 12 & 7 & 12 & 1 & \Rightarrow k_1 = 1,\, k_7 = 1 \text{ and } k_{12} = 2
\end{array}
$$

Figure 17: Example of binary expansions of a quad

We can now express each of the relevant quantities as linear functions of the k_j:

$$
\begin{aligned}
\operatorname{cost}(Q) &= k_1 + k_2 + \cdots + k_{15}, && (23)\\
|\nu_1 \vee \nu_2| &= k_4 + k_5 + k_6 + k_7 + k_8 + k_9 + k_{10} \\
&\quad + k_{11} + k_{12} + k_{13} + k_{14} + k_{15}, && (24)\\
|\nu_1 \vee \nu_3| &= k_2 + k_3 + k_6 + k_7 + k_8 + k_9 + k_{10} \\
&\quad + k_{11} + k_{12} + k_{13} + k_{14} + k_{15}, && (25)\\
|\nu_1 \vee \nu_4| &= k_1 + k_3 + k_5 + k_7 + k_8 + k_9 + k_{10} \\
&\quad + k_{11} + k_{12} + k_{13} + k_{14} + k_{15}, && (26)\\
|\nu_2 \vee \nu_3| &= k_2 + k_3 + k_4 + k_5 + k_6 + k_7 + k_{10} \\
&\quad + k_{11} + k_{12} + k_{13} + k_{14} + k_{15}, && (27)\\
|\nu_2 \vee \nu_4| &= k_1 + k_3 + k_4 + k_5 + k_6 + k_7 + k_9 \\
&\quad + k_{11} + k_{12} + k_{13} + k_{14} + k_{15}, && (28)\\
|\nu_3 \vee \nu_4| &= k_1 + k_2 + k_3 + k_5 + k_6 + k_7 + k_9 \\
&\quad + k_{10} + k_{11} + k_{13} + k_{14} + k_{15}, && (29)\\
\operatorname{sav}(\nu_1, \nu_3) &= k_{10} + k_{11} + k_{14} + k_{15}, && (30)\\
\operatorname{sav}(\nu_2, \nu_4) &= k_5 + k_7 + k_{13} + k_{15}, && (31)\\
\operatorname{sav}(\nu_1, \nu_4) &= k_9 + k_{11} + k_{13} + k_{15}, && (32)\\
\operatorname{sav}(\nu_2, \nu_3) &= k_6 + k_7 + k_{14} + k_{15}. && (33)
\end{aligned}
$$

Using the identities in (23)–(33), we can write the inequalities (17), (20), and (22) as follows:

$$\begin{aligned}\phi_Q &\leqslant k_1 + k_2 + k_3 + k_4 + 3/2 \cdot k_5 + 2k_6 + 3/2 \cdot k_7 + k_8 + 2k_9 \\ &\quad + 3/2 \cdot k_{10} + 3/2 \cdot k_{11} + k_{12} + 3/2 \cdot k_{13} + 3/2 \cdot k_{14} + k_{15}, \qquad (34)\\ \phi_Q &\leqslant k_1 + k_2 + 2k_3 + k_4 + k_5 + 3/2 \cdot k_6 + 3/2 \cdot k_7 + k_8 + 3/2 \cdot k_9 \\ &\quad + k_{10} + 3/2 \cdot k_{11} + 2k_{12} + 3/2 \cdot k_{13} + 3/2 \cdot k_{14} + k_{15}, \qquad (35)\\ \phi_Q &\leqslant k_1 + k_2 + 2k_3 + k_4 + 2k_5 + 1/2 \cdot k_6 + 3/2 \cdot k_7 + k_8 + 1/2 \cdot k_9 \\ &\quad + 2k_{10} + 3/2 \cdot k_{11} + 2k_{12} + 3/2 \cdot k_{13} + 3/2 \cdot k_{14} + k_{15}. \qquad (36)\end{aligned}$$

Recall that, all right-hand sides (17)-(22) are upper bounds on the contribution ϕ_Q. In fact, any convex combination of these six right-hand sides also gives an upperbound on the contribution of ϕ_Q. Multiplying (34) by $1/2$, (35) by $1/4$, and (36) by $1/4$, and then adding the three inequalities, we get, for each quad $Q \in O_0$:

$$\begin{aligned}\phi_Q &\leqslant k_1 + k_2 + 3/2 \cdot k_3 + k_4 + 3/2 \cdot k_5 + 3/2 \cdot k_6 + 3/2 \cdot k_7 + k_8 + 3/2 \cdot k_9 \\ &\quad + 3/2 \cdot k_{10} + 3/2 \cdot k_{11} + 3/2 \cdot k_{12} + 3/2 \cdot k_{13} + 3/2 \cdot k_{14} + k_{15} \\ &= 3/2 \sum_{j=1}^{15} k_j - 1/2 \cdot k_1 - 1/2 \cdot k_2 - 1/2 \cdot k_4 - 1/2 \cdot k_8 - 1/2 \cdot k_{15} \\ &= 3/2 \cdot \text{cost}(Q) - 1/2 \cdot k_1 - 1/2 \cdot k_2 - 1/2 \cdot k_4 - 1/2 \cdot k_8 - 1/2 \cdot k_{15} \\ &\leqslant 3/2 \cdot \text{cost}(Q). \qquad (37)\end{aligned}$$

Thus, (9), (14), and (37) show that indeed:

$$\phi_Q \leqslant 3/2 \cdot \text{cost}(Q) \text{ for each } Q \text{ from the optimum solution.}$$

Proving that $\sum_Q \phi_Q = \text{cost}(M^{A1}) - (S_1 + 1/2 \cdot S_2)$

Since a quad contains either zero, one, or two lucky pairs, it follows that the expressions for ϕ_Q given in (8), (10), (17)-(22) contain the terms that

jointly sum up to $\text{cost}(M^{A1})$. Additionally, it is not difficult to verify that the construction of the graph H is such that the savings on the edges of H (as defined in (8), (10), (17)-(22)) sum up to $S_1 + 1/2 \cdot S_2$.

The proof is complete. □

4.3.2 *Analysis for PQ(#1 = 2)*

Lemma 4.5. *The worst-case ratio of algorithm A for PQ(#1 = 2) is at most 4/3.*

Proof. Recall that an instance of PQ(#1 = 2) is a multi-graph G with $4n$ edges. We assume that the algorithm chooses in the first phase a matching that matches as many pairs of duplicates as possible (see Lemma 4.3).

Our analysis follows the structure of the proof of Lemma 4.4, but we obtain a better bound on the ratio between ϕ_Q and $\text{cost}(Q)$ by exploiting the restricted set of configurations that are possible for a quad Q in the optimal solution if every vector in the given instance of the problem has exactly two ones.

By specifying two edge pairs in each quad from the optimal solution, we obtain a matching M^{A1} whose cost is an upper bound on the cost of the matching M^1 computed by the algorithm in the first phase, i.e.

$$\text{cost}(M^1) \leqslant \text{cost}(M^{A1}), \text{ for any possible choice of } M^{A1}. \quad (38)$$

Furthermore, we again construct an auxiliary graph $H = (V', E_1 \cup E_2)$ that represents potential matches between vector pairs in M^1 with corresponding savings that algorithm A could make in the second phase. The weight $w(e)$ of each edge e in the graph H represents the savings that algorithm A would realize in the second phase if it were to match the vector pairs that are the endpoints of e. The graph H is again bipartite (we will use the concept of good partitions introduced in Section 4.3.1 to ensure this) and has maximum degree 2, and the edges in E_1 connect

vertices of degree 1 and the edges in E_2 connect vertices of degree 2. Letting S_1 denote the total weight of E_1, and S_2 the total weight of E_2, it follows from Claim 1 that the maximum-weight matching M^{A2} in H satisfies $\text{savings}(M^{A2}) \geqslant S_1 + 1/2 \cdot S_2$, and hence the weight of the matching M^2 that algorithm A finds in the second phase satisfies:

$$\text{savings}(M^2) \geqslant \text{savings}(M^{A2}) \geqslant S_1 + 1/2 \cdot S_2. \tag{39}$$

Thus, we again have the relationship

$$A(I) \leqslant \text{cost}(M^{A1}) - (S_1 + 1/2 \cdot S_2) \tag{40}$$

and distribute the value on the right-hand side over the quads of the optimal solution, with each quad Q receiving a share ϕ_Q that is referred to as the contribution reserved for Q. We will show that $\phi_Q \leqslant 4/3 \cdot \text{cost}(Q)$ holds for all quads Q of the optimal solution, implying that:

$$\begin{aligned} A(I) &\leqslant \text{cost}(M^{A1}) - (S_1 + 1/2 \cdot S_2) = \sum_Q \phi_Q \\ &\leqslant \sum_Q 4/3 \cdot \text{cost}(Q) = 4/3 \cdot OPT(I) \end{aligned}$$

Thus, correctness hinges upon proving that

(i) $\phi_Q \leqslant 4/3 \cdot \text{cost}(Q)$ for each Q from the optimum solution,

(ii) the graph H that we construct is bipartite and has maximum degree 2, and

(iii) $\sum_Q \phi_Q = \text{cost}(M^{A1}) - (S_1 + 1/2 \cdot S_2)$.

Proving that $\phi_Q \leqslant 4/3 \cdot \text{cost}(Q)$ for each Q from the optimum solution

Note that a quad contains four edges of multi-graph G and its cost equals the number of vertices in the subgraph induced by these four

edges. Thus, in this proof we write quad Q from the optimum solution as a set of four edges of G, i.e., $Q = (e_1, e_2, e_3, e_4)$. We distinguish the following cases depending on the value of cost(Q):



Figure 18: Quads with cost(Q) = 2); 1 and 2 are nodes in G

- cost(Q) = 2. This means all four edges are identical, see Figure 18. As we may assume that the algorithm, in the first phase, matches as many duplicate vectors (edges of G) as possible (Lemma 4.3), the algorithm matches $p_1 = \{e_1, e_2\}$ and $p_2 = \{e_3, e_4\}$. We select both p_1 and p_2 to be part of matching M^{A1}, with total cost $2+2 = 4$.

  If the algorithm, in the second phase, matches p_1 and p_2, the algorithm will make a savings of 2. Thus, we add edge (p_1, p_2) with savings 2 to edge set S_1 in H, as both p_1 and p_2 do not appear in any other quads. Hence,

$$\phi_Q \leqslant \text{cost}(p_1) + \text{cost}(p_2) - \text{sav}(p_1, p_2) = 2 \leqslant 4/3 \cdot \text{cost}(Q).$$

- cost(Q) = 3. The edges in G belonging to Q can have one of the following three structures, see Figure 19:

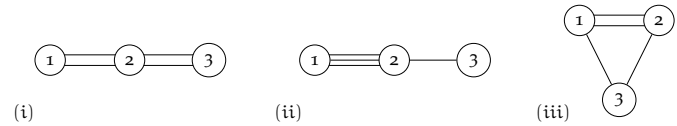


Figure 19: Quads with cost(Q) = 3; 1,2 and 3 are nodes in G

(i) If quad Q contains $e_1 = e_2 = (1,2)$ and $e_3 = e_4 = (2,3)$, we select both $p_1 = \{(1,2),(1,2)\}$ and $p_2 = \{(2,3),(2,3)\}$ to be part of matching M^{A1}, with a total cost of $2+2 = 4$. Furthermore,

we add the edge (p_1, p_2) with savings 1 to the edge set E_1 of H. Hence,

$$\phi_Q \leqslant \text{cost}(p_1) + \text{cost}(p_2) - \text{sav}(p_1, p_2) = 4 - 1 \leqslant 4/3 \cdot \text{cost}(Q).$$

(ii) If quad Q contains $e_1 = e_2 = e_3 = (1,2)$ and $e_4 = (2,3)$, we may assume that the algorithm has matched $p_1 = \{(1,2),(1,2)\}$ in the first phase. Furthermore, we select p_1 and $p_2 = \{e_3, e_4\}$ to be part of matching M^{A1}, with total cost $2 + 3 = 5$.

In a worst case scenario, p_2 is not contained in M^1, hence the algorithm has matched e_3 to another vector (edge in G), say x_3, and e_4 to say x_4. We define that $p_3 = \{e_3, x_3\}$ and $p_4 = \{e_4, x_4\}$, which are nodes in the auxiliary graph H. We add edges (p_1, p_3) and (p_1, p_4) to edge set E_2 of H, with total savings $2 + 1 = 3$. Hence,

$$\begin{aligned}\phi_Q &\leqslant \text{cost}(p_1) + \text{cost}(p_2) - 1/2\Big(\text{sav}(p_1, p_3) + \text{sav}(p_1, p_4)\Big) \\ &= 5 - 3/2 \leqslant 4/3 \cdot \text{cost}(Q).\end{aligned}$$

(iii) If quad Q contains $e_1 = e_2 = (1,2)$, $e_3 = (1,3)$ and $e_4 = (2,3)$, we use the same argument as above. We may assume that the algorithm has matched $p_1 = \{(1,2),(1,2)\}$ in the first phase and select p_1 and $p_2 = \{e_3, e_4\}$ to be part of matching M^{A1}, with total cost $2 + 3 = 5$.

In a worst case scenario, p_2 is not contained in M^1, hence the algorithm has matched e_3 to say x_3, and e_4 to say x_4. We define $p_3 = \{e_3, x_3\}$ and $p_4 = \{e_4, x_4\}$, which are nodes in auxiliary graph H. We add edges (p_1, p_3) and (p_1, p_4) to edge set E_2 of H, with total savings $1 + 1 = 2$. Hence,

$$\begin{aligned}\phi_Q &\leqslant \text{cost}(p_1) + \text{cost}(p_2) - 1/2\Big(\text{sav}(p_1, p_3) + \text{sav}(p_1, p_4)\Big) \\ &= 5 - 1 = 4/3 \cdot \text{cost}(Q).\end{aligned}$$

- $\text{cost}(Q) = 4$. The edges in G belonging to Q can have one of the following six structures, see Figure 20:

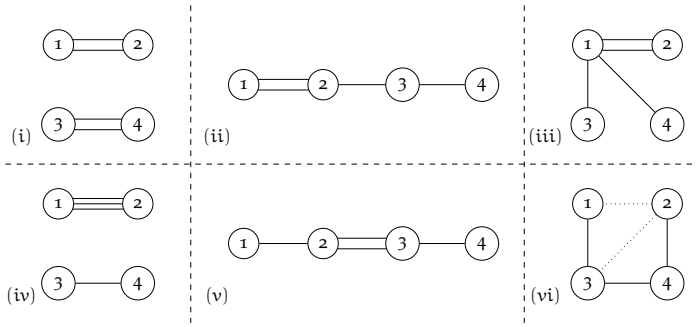


Figure 20: Quads with $\text{cost}(Q) = 4$; 1, 2, 3 and 4 are nodes in G

(i) If quad Q contains $e_1 = e_2 = (1,2)$ and $e_3 = e_4 = (3,4)$, then we select both $p_1 = \{(1,2),(1,2)\}$ and $p_2 = \{(3,4),(3,4)\}$ to be part of matching M^{A1}, with a total cost of $2+2 = 4$. We add the edge (p_1, p_2) with savings 0 to the edge set E_1 of H. Hence,

$$\phi_Q \leqslant \text{cost}(p_1) + \text{cost}(p_2) - \text{sav}(p_1, p_2) = 4 \leqslant 4/3 \cdot \text{cost}(Q).$$

(ii)&(iii) If the graph induced by quad Q contains two identical edges, e_1 and e_2 and two adjacent edges, e_3 and e_4, then we may assume that algorithm A has matched $p_1 = \{e_1, e_2\}$ in the first phase. We select p_1 and $p_2 = \{e_3, e_4\}$ to be part of matching M^{A1}, with a total cost of $2+3 = 5$.

In a worst case scenario, p_2 is not contained in M^1, hence the algorithm has matched e_3 to say x_3 and e_4 to say x_4. Let $p_3 = (e_3, x_3)$ and $p_4 = (e_4, x_4)$, and add the edges (p_1, p_3)

and (p_1, p_4) with total savings at least $1 + 0 = 1$ to the edge set E_2 of H. Hence,

$$\begin{aligned}\phi_Q &\leqslant \text{cost}(p_1) + \text{cost}(p_2) - 1/2\Big(\text{sav}(p_1, p_3) + \text{sav}(p_1, p_4)\Big)\\ &= 5 - 1/2 \leqslant 4/3 \cdot \text{cost}(Q).\end{aligned}$$

(iv)&(v) If the graph induced by quad Q contains two identical edges, e_1 and e_2 and two non-adjacent edges, e_3 and e_4, then we may assume that algorithm A has matched $p_1 = \{e_1, e_2\}$ in the first phase. We select $p_1 = \{e_1, e_2\}$ and $p_2 = \{e_3, e_4\}$ to be part of matching M^{A1}, with a total cost of $2 + 4 = 6$.

In a worst case scenario, p_2 is not contained in M^1, hence the algorithm has matched e_3 to another vector (edge in G), say x_3, and e_4 to say x_4. We define that $p_3 = \{e_3, x_3\}$ and $p_4 = \{e_4, x_4\}$, which are nodes in auxiliary graph H. We add edges (p_1, p_3) and (p_1, p_4) to edge set E_2 of H. Either e_3 is a duplicate of e_1 (case (iv)), in which case the total saving is $2 + 0 = 2$. Or both edges are adjacent to both e_1 and e_2 (case (v)), in which case the total saving is $1 + 1 = 2$. Hence,

$$\begin{aligned}\phi_Q &\leqslant \text{cost}(p_1) + \text{cost}(p_2) - 1/2\Big(\text{sav}(p_1, p_3) + \text{sav}(p_1, p_4)\Big)\\ &\leqslant 6 - 1 \leqslant 4/3 \cdot \text{cost}(Q).\end{aligned}$$

(vi) Suppose quad Q contains $e_1 = (1,2)$, $e_2 = (1,3)$, $e_3 = (2,4)$ and $e_4 = (2,3)$ (if $e_1 = (2,3)$ we use the same argument). We select $p_1 = \{e_1, e_2\}$ and $p_2 = \{e_3, e_4\}$ to be part of matching M^{A1}, with total cost $3 + 3 = 6$.

In a worst case scenario, no edge pair of G in Q is contained in M^1, hence we say the algorithm has matched e_i to x_i and define $p_{e_i} = \{e_i, x_i\}$ for $i \in \{1, \ldots, 4\}$. Note that each of the p_{e_i} is a node in auxiliary graph H. If the good partition associated with Q (as defined in Section 4.3.1) is $\{\{p_{e_1}, p_{e_2}\}, \{p_{e_3}, p_{e_4}\}\}$, we add edges (p_{e_1}, p_{e_3}) and (p_{e_2}, p_{e_4})

to edge set E_2 of H, with total savings at least $1 + 1 = 2$. Otherwise, we add edges (p_{e_1}, p_{e_2}) and (p_{e_3}, p_{e_4}) to edge set E_2, with total savings at least $1 + 1 = 2$. Hence,

$$\phi_Q \leqslant \text{cost}(p_1) + \text{cost}(p_2) - 1 = 5 < 4/3 \cdot \text{cost}(Q).$$

- $\text{cost}(Q) = 5$. If Q contains two identical edges, say e_1 and e_2, we select $p_1 = \{e_1, e_2\}$ and $p_2 = \{e_3, e_4\}$ to be part of M^{A1}, with a total cost of at most $2 + 4 = 6$. We can assume that the algorithm has matched (e_1, e_2) in M^1. In the worst case, the algorithm has not matched e_3 to e_4 in M^1. Let $p_{e_3} = \{e_3, x_3\}$ and $p_{e_4} = \{e_4, x_4\}$ be the matched pairs in M^1 that contain e_3 and e_4, respectively. We add the edges (p_1, p_{e_3}) and (p_1, p_{e_4}) to the edge set E_2 of H. Even without taking the savings of those edges into account, we have

  $$\phi_Q \leqslant \text{cost}(p_1) + \text{cost}(p_2) = 6 < 4/3 \cdot \text{cost}(Q).$$

  Assume now that Q does not contain any two identical edges. Then the graph induced by $Q = (e_1, e_2, e_3, e_4)$ must fall into one of the four following cases, see Figure 21:

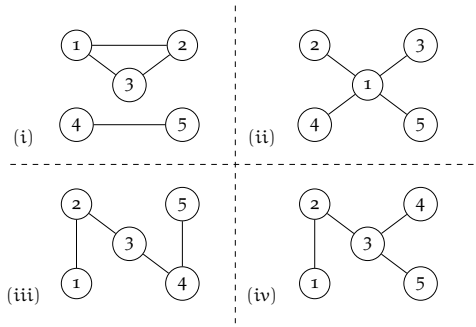


Figure 21: Quads with $\text{cost}(Q) = 5$; 1, 2, 3, 4 and 5 are nodes in G

(i) Q is a triangle of edges e_1, e_2, e_3 plus a disjoint edge e_4. We select $p_1 = \{e_1, e_2\}$ and $p_2 = \{e_3, e_4\}$ to be part of M^{A1},

with a total cost of $3+4=7$. In the worst case, Q does not contain a lucky pair. For $1 \leqslant i \leqslant 4$, let $p_{e_i} = \{e_i, x_i\}$ be the pair of matched edges in M^1 that includes e_i. If the good partition associated with Q is $\{\{p_{e_1}, p_{e_2}\}, \{p_{e_3}, p_{e_4}\}\}$, we add the edges (p_{e_1}, p_{e_3}) and (p_{e_2}, p_{e_4}) to E_2 with savings at least $1+0=1$. Otherwise, add the edges (p_{e_1}, p_{e_2}) and (p_{e_3}, p_{e_4}) to E_2, again with savings at least $1+0=1$. Hence,

$$\phi_Q \leqslant \text{cost}(p_1) + \text{cost}(p_2) - 1/2 = 7 - 1/2 < 4/3 \cdot \text{cost}(Q).$$

(ii) Q is a star (i.e., all four edges of Q are incident with the same same vertex). We select $p_1 = \{e_1, e_2\}$ and $p_2 = \{e_3, e_4\}$ to be part of M^{A1}, with a total cost of $3+3=6$. In the worst case, Q does not contain a lucky pair. For $1 \leqslant i \leqslant 4$, let $p_{e_i} = \{e_i, x_i\}$ be the pair of matched edges in M^1 that includes e_i. If the good partition associated with Q is $\{\{p_{e_1}, p_{e_2}\}, \{p_{e_3}, p_{e_4}\}\}$, add the edges (p_{e_1}, p_{e_3}) and (p_{e_2}, p_{e_4}) to E_2, with savings at least $1+1=2$. Otherwise, add the edges (p_{e_1}, p_{e_2}) and (p_{e_3}, p_{e_4}) to E_2, again with savings at least $1+1=2$. Hence,

$$\phi_Q \leqslant \text{cost}(p_1) + \text{cost}(p_2) - 1 = 5 < 4/3 \cdot \text{cost}(Q).$$

(iii) Q is a path of four edges. Assume that the edges appear on the path in the order (e_1, e_2, e_3, e_4). We select $p_1 = \{e_1, e_2\}$ and $p_2 = \{e_3, e_4\}$ to be part of M^{A1}, with a total cost of $3+3=6$. In the worst case, Q does not contain a lucky pair. For $1 \leqslant i \leqslant 4$, let $p_{e_i} = \{e_i, x_i\}$ be the pair of matched edges in M^1 that includes e_i. If the good partition associated with Q is $\{\{p_{e_1}, p_{e_2}\}, \{p_{e_3}, p_{e_4}\}\}$, add the edges (p_{e_1}, p_{e_4}) and (p_{e_2}, p_{e_3}) to E_2, with savings at least $0+1=1$. Otherwise, add the edges (p_{e_1}, p_{e_2}) and (p_{e_3}, p_{e_4}) to E_2, with savings at least $1+1=2$. Hence,

$$\phi_Q \leqslant \text{cost}(p_1) + \text{cost}(p_2) - 1/2 = 6 - 1/2 < 4/3 \cdot \text{cost}(Q).$$

(iv) Q is a tree of diameter 3. Assume that $e_1 = (1,2)$, $e_2 = (2,3)$, $e_3 = (3,4)$, $e_4 = (3,5)$. We select $p_1 = \{e_1, e_2\}$ and $p_2 = \{e_3, e_4\}$ to be part of M^{A1}, with a total cost of $3+3=6$. In the worst case, Q does not contain a lucky pair. For $1 \leqslant i \leqslant 4$, let $p_{e_i} = \{e_i, x_i\}$ be the pair of matched edges in M^1 that includes e_i. If the good partition associated with Q is $\{\{p_{e_1}, p_{e_2}\}, \{p_{e_3}, p_{e_4}\}\}$, add the edges (p_{e_1}, p_{e_4}) and (p_{e_2}, p_{e_3}) to E_2, with savings at least $0+1=1$. Otherwise, add the edges (p_{e_1}, p_{e_2}) and (p_{e_3}, p_{e_4}) to E_2, with savings at least $1+1=2$. Hence,

$$\phi_Q \leqslant \text{cost}(p_1) + \text{cost}(p_2) - 1/2 = 6 - 1/2 < 4/3 \cdot \text{cost}(Q).$$

- $\text{cost}(Q) \geqslant 6$. If we partition Q into two edge pairs p_1 and p_2 for matching M^{A1}, their total cost will be at most $4+4=8$. Even if the savings of the edges that we add to the edge set E_2 of H (while taking into account the good partition associated with Q to ensure that H is bipartite, of course) are zero, we have

$$\phi_Q \leqslant \text{cost}(p_1) + \text{cost}(p_2) = 8 \leqslant 4/3 \cdot \text{cost}(Q).$$

Proving that H *is a simple bipartite graph with maximum degree* 2

An edge has been added to the edge set E_1 of H when considering a quad Q only if Q is a quad with two lucky pairs, and hence those pairs indeed become vertices of degree 1 in H. If $p = (e_1, e_2)$ is a lucky pair that is in a quad Q with two other edges e_3, e_4 that do not form a lucky pair, it becomes the endpoint of exactly two edges in E_2: The edges (p, p_3) and (p, p_4), where p_3 is the pair of edges containing e_3 that is matched in M^1, and p_4 is the pair of edges containing e_4 that is matched in M^1. Finally, every edge pair $p = (e_1, e_2)$ that has been matched by algorithm A in M^1 and is not a lucky pair becomes the

endpoint of exactly two edges in E_2: An edge (p, p') added for quad Q_1 and an edge (p, p'') added for quad Q_2, where Q_1 is the optimal quad containing e_1 and Q_2 the optimal quad containing e_2. This shows that H is a graph with maximum degree 2 in which each edge connects two vertices of the same degree. Furthermore, by ensuring for each quad Q that the edges added to E_2 are compatible with the good partition associated with Q if Q does not have a lucky pair, we have ensured that H is simple and bipartite (recall that the cycles in H correspond to Eulerian cycles of the connected components of K, which have an even number of edges, as discussed in Section 4.3.1).

Proving that $\sum_Q \phi_Q = \text{cost}(M^{A1}) - (S_1 + 1/2 \cdot S_2)$

For each quad Q we have selected two pairs of edges in Q for inclusion in M^{A1}, so the costs of those edge pairs clearly add up to $\text{cost}(M^{A1})$. Furthermore, each edge of H was added to H by some quad Q. Furthermore, no two quads could have added the same edge to H (as H is a simple graph), so the sum of the savings of the added edges is indeed $S_1 + 1/2 \cdot S_2$.

The proof is complete. □

4.3.3 *Analysis for PQ(#1 = 2, distinct)*

Lemma 4.6. *The worst-case ratio of algorithm A for PQ(#1 = 2, distinct) is at most 13/10.*

Proof. Recall that an instance of PQ(#1 = 2, distinct) is nothing else but a simple graph F with $4n$ edges. Note that the cost of every optimal quad is at least 4 since 4 edges in a simple graph touch at least 4 different vertices. Hence we can repeat the arguments in the proof for Lemma 4.3.2 for quads of cost 4 and higher.

Then, we get the following results:

- For $\text{cost}(Q) = 4$, we have $\phi_Q \leqslant 5 = 5/4 \cdot \text{cost}(Q) < 13/10 \cdot \text{cost}(Q)$,
- For $\text{cost}(Q) = 5$, we have $\phi_Q \leqslant 6.5 = 6.5/5 \cdot \text{cost}(Q) = 13/10 \cdot \text{cost}(Q)$,
- For $\text{cost}(Q) = 6$, note that the average degree of the subgraph of G induced by Q is $8/6 > 1$, so there must exist a vertex of degree at least 2. This means that Q contains two adjacent edges e_1 and e_2. Denote the remaining edges by e_3 and e_4. We can select the pairs $p_1 = \{e_1, e_2\}$ and $p_2 = \{e_3, e_4\}$ to be part of M^{A1}, giving $\phi_Q \leqslant \text{cost}(p_1) + \text{cost}(p_2) \leqslant 3 + 4 = 7 = 7/6 \cdot \text{cost}(Q) < 13/10 \cdot \text{cost}(Q)$.
- For $\text{cost}(Q) \geqslant 7$, we have $\phi_Q \leqslant 8 \leqslant 8/7 \cdot \text{cost}(Q) < 13/10 \cdot \text{cost}(Q)$.

□

4.3.4 *Analysis for PQ(#1 = 2, distinct, connected)*

Lemma 4.7. *The worst-case ratio of algorithm A for problem PQ(#1 = 2, distinct, connected) is at most 5/4.*

Proof. Recall that an instance of PQ(#1 = 2, distinct, connected) is nothing else but a simple, connected graph F with $4n$ edges. Note that the cost of every optimal quad is at least 4 since 4 edges in a simple graph touch at least 4 different vertices. Hence, $OPT \geqslant 4n$. Furthermore, if we can show that there are z quads in the optimal solution that have cost at least 5, we get that $OPT \geqslant 4(n - z) + 5z = 4n + z$.

Observation 4.3. $\text{cost}(M) = 6n$.

Proof. Algorithm A produces, in the first phase, a matching M^1 of cost $6n$ as each of the $2n$ pairs of matched edges has cost 3. This follows from the fact that the line graph of a connected graph with an even number of edges admits a perfect matching (Jünger et al., 1985; Dong et al., 2013), and hence the cost of M^1 is minimized by a matching in which adjacent edges are paired. □

Let $p_1, \ldots, p_{2n}$ be the pairs corresponding to M^1. Consider an auxiliary graph H with vertex set $V' = \{p_1, \ldots, p_{2n}\}$ in which an edge is added between p_i and p_j if p_i and p_j have at least one common vertex (implying that matching p_i to p_j in the matching M^2 that A computes in the second phase would create a saving of at least one). Note that H is connected as F is connected. Let μ be the size of a maximum matching in H, $1 \leqslant \mu \leqslant n$. Note that the maximum matching of H can be extended to a perfect matching of V' that makes savings at least μ. Therefore, we have

$$A(I) \leqslant 6n - \mu.$$

If H contains a perfect matching, we have $\mu = n$ and hence $A(I) \leqslant 5n$, implying $A(I)/OPT(I) \leqslant 5n/(4n) = 5/4$. It remains to consider the case $\mu < n$.

If a maximum matching in H has size $\mu < n$, the number of unmatched vertices is $2n - 2\mu$. We will show that the optimal solution then contains at least $n - \mu$ quads with cost at least 5, and hence we have $OPT(I) \geqslant 4n + (n - \mu) = 5n - \mu$. Therefore,

$$\frac{A(I)}{OPT(I)} \leqslant \frac{6n - \mu}{5n - \mu} \leqslant \frac{5}{4}$$

where the last inequality follows because $(6n - \mu)/(5n - \mu)$ is maximized if μ takes its maximum possible value, $\mu = n$.

It remains to show that the optimal solution contains at least $n - \mu$ quads with cost at least 5. Recall that a maximum matching in H leaves $2n - 2\mu$ vertices unmatched. By the Tutte-Berge formula (Berge, 1958), the number of unmatched vertices of a maximum matching in H is equal to

$$\max_{X \subseteq V'} (\mathrm{odd}(H - X) - |X|)$$

where $\mathrm{odd}(H - X)$ is the number of connected components of $H - X$ that have an odd number of vertices ($H - X$ is the graph that results when the nodes in X (and the edges adjacent to it) are removed from H). Hence, there exists a set $X \subseteq P$ such that $\mathrm{odd}(H - X) - |X| = 2n - 2\mu$. Let

$\delta = \text{odd}(H - X)$, and let $O_1, O_2, \ldots, O_d$ denote the d odd components of $H - X$. We have:

$$2n - 2\mu = \delta - |X|$$

For a subgraph S of H, let $E_F(S)$ denote the set of edges of F that are contained in the edge pairs that form the vertex set of S (recall that the vertices of H are pairs of edges from F). Note that $|E_F(O_i)| \bmod 4 = 2$ for $1 \leqslant i \leqslant \delta$ as O_i contains an odd number of edge pairs. Therefore, each $E_F(O_i)$ contains at least two edges that are contained in optimal quads that do not only contain edges from $E_F(O_i)$. If such a quad contains three edges from $E_F(O_i)$, note that there must be at least one other optimal quad that contains at most three edges from $E_F(O_i)$ as $(|E_F(O_i)| - 3) \bmod 4 = 3$.

For each optimal quad that contains one or two edges from $E_F(O_i)$, define these one or two edges to be *special* edges. For each optimal quad that contains three edges from $E_F(O_i)$, select one of these three edges arbitrarily and define it to be a *special* edge. There are at least two special edges in each $E_F(O_i)$, $1 \leqslant i \leqslant \delta$, and hence at least 2δ special edges in total. More precisely, we refer to these special edges as the edge set SE, and partition it into two subsets: those special edges occurring in a quad with cost 4 (the set SE4), and those special edges occurring in a quad with cost at least 5 (the set SE5). Clearly:

$$2\delta \leqslant |SE4| + |SE5|. \tag{41}$$

Consider a quad with cost 4 from the optimum solution. It consists of four edges of F. Since F is a connected simple graph there are only two possible subgraphs induced Q, as depicted in Figure 22. These four edges can be in the sets $E_F(O_i)$, the set $E_F(X)$, and the set $E_F(C)$. We now define types of quads of cost 4 depending on how many edges are in which set.

Note that an edge from $E_F(O_i)$ cannot be incident to the same vertex as an edge from $E_F(O_j)$ for $j \neq i$ because otherwise H would contain an edge between O_i and O_j. Similarly, an edge from $E_F(O_i)$ cannot be

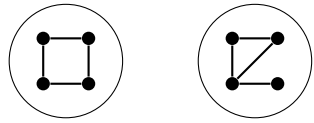


Figure 22: Quads with $\text{cost}(Q) = 4$

incident to the same vertex as an edge from $E_F(C)$ where C is an even component of $H - X$. The only edges that can share endpoints with edges in $E_F(O_i)$ are those in $E_F(X)$.

We tabulate the different types of quads with cost 4 in Table 10.

Type of quad	Number of edges in $E_F(O_i)$	$E_F(X)$	$E_F(C)$	Cost	Number of special edges
1	3	1		4	1
2	2	2		4	2
3	1,1	2		4	2
4	1	2	1	4	1
5	1	3		4	1

Table 10: Overview of different types of quads with cost 4, containing at least 1 edge from $E_F(O_i)$

Thus, a quad with cost 4 with a special edge must be of type 1,2,3,4 or 5. For each of these types, the number of edges from $E_F(X)$ is at least the number of special edges in the quad. Thus,

$$|E_F(X)| \geqslant |SE4|. \tag{42}$$

Further, since $|E_F(X)| = 2|X|$, it follows from (42) and (41):

$$SE5 \geqslant 2\delta - 2|X|.$$

Thus, the number of quads with cost at least 5 is at least $\frac{2\delta - 2|X|}{4} = 1/2(\delta - |X|) = n - \mu$.

As shown before, this implies that $OPT(I) \geqslant 5n - \mu$, hence

$$\frac{A(I)}{OPT(I)} \leqslant \frac{6n - \mu}{5n - \mu} \leqslant \frac{5}{4}.$$

□

4.4 LOWER BOUND APPROXIMATION GUARANTEE

In this section, we provide instances that, together with the results in the previous section, lead to the bounds claimed.

4.4.1 *Instances of PQ(#1 ∈ {1, 2})*

Consider the instance I consisting of the following 8 vectors, $v_1, \ldots, v_8$:

$$\begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ 0 \\ 1 \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \end{pmatrix}, \begin{pmatrix} 1 \\ 1 \\ 0 \\ 0 \end{pmatrix}, \begin{pmatrix} 1 \\ 1 \\ 0 \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ 0 \\ 1 \\ 1 \end{pmatrix}, \begin{pmatrix} 0 \\ 0 \\ 1 \\ 1 \end{pmatrix}.$$

Since each vector contains either one or two 1's, this is an instance of PQ(#1 ∈ {1, 2}). Clearly, the optimum solution consists of the quads $\{v_1, v_2, v_5, v_6\}$ and $\{v_3, v_4, v_7, v_8\}$, with a total cost of $OPT(I) = 4$; algorithm A however, may find, as an optimum matching in the first phase, the pairs $\{v_1, v_3\}$, $\{v_2, v_4\}$, $\{v_5, v_6\}$ and $\{v_7, v_8\}$, leading to a final solution with cost $A(I) = 6$. Thus, we arrive at the following observation.

Observation 4.4. *For the instance depicted above we have,* $\text{cost}(A) = 3/2 \cdot \text{OPT}$.

Theorems 4.2 and 4.3 now follow from Lemma 4.4 and Observation 4.4.

As a remark, if we would allow all-zero vectors in the input (which we do not allow in PQ(#1 $\in \{1,2\}$)), we can get an even smaller example with ratio $3/2$. Let instance I consist of the following 8 vectors:

$$\begin{pmatrix}1\\0\end{pmatrix},\begin{pmatrix}1\\0\end{pmatrix},\begin{pmatrix}1\\0\end{pmatrix},\begin{pmatrix}0\\0\end{pmatrix},\begin{pmatrix}0\\1\end{pmatrix},\begin{pmatrix}0\\1\end{pmatrix},\begin{pmatrix}0\\1\end{pmatrix},\begin{pmatrix}0\\0\end{pmatrix}.$$

The optimal cost is 2; algorithm A however, may match the two all-zero vectors in the first phase and get a solution of cost 3.

Note that both these instances can be generalized to larger instances with the same lower bound. For example, we can double the size of the above instance above as follows. Instead of 8 vectors we will consider 16 vectors each with dimension 4. Vector v'_i with $i \in \{1,\ldots,8\}$ will have components 1 and 2 identical to components 1 and 2 of vector v_i of the instance above and a 0 in components 3 and 4. Vector v'_{8+i} with $i \in \{1,\ldots,8\}$ will have a 0 in components 1 and 2 and components 3 and 4 will be identical to components 1 and 2 of vector v_i of the instance above. Likewise we can create instances with size $s = 0 \mod 8$ that give the same lower bound.

4.4.2 *Instances of PQ(#1 = 2, connected)*

Consider the instance I consisting of the following 8 vectors, $v_1,\ldots,v_8$.

$$\begin{pmatrix}1\\1\\0\\0\\0\end{pmatrix},\begin{pmatrix}1\\1\\0\\0\\0\end{pmatrix},\begin{pmatrix}1\\0\\1\\0\\0\end{pmatrix},\begin{pmatrix}0\\1\\1\\0\\0\end{pmatrix},\begin{pmatrix}0\\0\\1\\1\\0\end{pmatrix},\begin{pmatrix}0\\0\\1\\0\\1\end{pmatrix},\begin{pmatrix}0\\0\\0\\1\\1\end{pmatrix},\begin{pmatrix}0\\0\\0\\1\\1\end{pmatrix},$$

Since each vector contains two 1's and the induced graph is connected, this this is an instance of PQ(#1 $= 2$, connected). This instance can be represented by the graph shown in Figure 23.

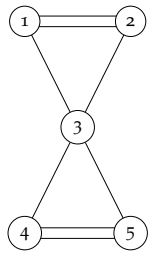


Figure 23: An instance of PQ(#1 = 2, connected)

The optimal solution for this instance has cost 6, with the two quads

$$\{v_1, v_2, v_3, v_4\} = \{(1,2), (1,2), (1,3), (2,3)\},$$

$$\{v_5, v_6, v_7, v_8\} = \{(3,4), (3,5), (4,5), (4,5)\}.$$

The algorithm may in the first phase construct a matching with cost 10 consisting of the following pairs:

$$\{v_1, v_2\} = \{(1,2), (1,2)\},\ \{v_3, v_5\} = \{(1,3), (3,4)\},$$

$$\{v_4, v_6\} = \{(2,3), (3,5)\},\ \{v_7, v_8\} = \{(4,5), (4,5)\}.$$

Any two pairs share at most 1 node. Hence, the total savings that can be made in the second matching are at most 2, so by Corollary 4.1 we have $A(I) \geqslant 8$. Hence, the worst-case ratio of A is at least $8/6 = 4/3$.

Observation 4.5. *For the instance depicted in Figure 23 we have,* $\text{cost}(A) = 4/3 \cdot \text{OPT}$.

Theorems 4.4 and 4.5 now follow from Lemma 4.5 and Observation 4.5.

Note that for PQ(#1 = 2, connected) this instance can be generalized to larger instances with the same lower bound as well. For example, we can double the size of the above instance above as follows. We create a copy of the above graph with nodes $1', \ldots 5'$. We 'connect' these two

graphs by saying that node 3 and node $3'$ are the same node. Hence, node 3 will also be adjacent to nodes $1'$, $2'$, $4'$ and $5'$.

By adding copies of the graph as described above we can create larger instances with the same lower bound.

4.4.3 *Instances of PQ(#1 = 2, distinct, connected)*

Consider the instance I consisting of the following 8 vectors $v_1, \ldots, v_8$.

$$\begin{pmatrix}1\\1\\0\\0\\0\\0\\0\end{pmatrix}, \begin{pmatrix}1\\0\\1\\0\\0\\0\\0\end{pmatrix}, \begin{pmatrix}0\\1\\0\\1\\0\\0\\0\end{pmatrix}, \begin{pmatrix}0\\0\\1\\1\\0\\0\\0\end{pmatrix}, \begin{pmatrix}0\\0\\0\\1\\1\\0\\0\end{pmatrix}, \begin{pmatrix}0\\0\\0\\1\\0\\1\\0\end{pmatrix}, \begin{pmatrix}0\\0\\0\\0\\1\\0\\1\end{pmatrix}, \begin{pmatrix}0\\0\\0\\0\\0\\1\\1\end{pmatrix}.$$

Since each vector contains two 1's, the vectors are pairwise distinct, and the induced graph is connected, this is an instance of PQ(#1 = 2}, distinct, connected). The instance can be represented by the graph shown in Figure 24. The optimal solution for this instance has cost 8,

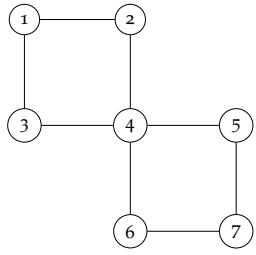


Figure 24: An instance of PQ(#1 = 2, distinct, connected)

with the two quads

$$\{v_1, v_2, v_3, v_4\} = \{(1,2), (1,3), (2,4), (3,4)\},$$

$$\{v_5, v_6, v_7, v_8\} = \{(4,5), (4,6), (5,7), (6,7)\}.$$

Algorithm A may, in the first phase, construct a matching with cost 12 consisting of the following pairs:

$$\{v_1, v_2\} = \{(1,2), (1,3)\},\ \{v_3, v_5\} = \{(2,4), (4,5)\},$$

$$\{v_4, v_6\} = \{(3,4), (4,6)\},\ \{v_7, v_8\} = \{(5,7), (6,7)\}.$$

Any two pairs share at most 1 node. Hence, the total savings that can be made in the second matching are at most 2, so by Corollary 4.1 we have $A(I) \geqslant 10$. Hence, the worst-case ratio of A is at least $10/8 = 5/4$.

Observation 4.6. *For the instance depicted in Figure 24 we have,* $\text{cost}(A) = 5/4 \cdot \text{OPT}$.

Theorem 4.6 now follows from Lemma 4.6 and Observation 4.6, and Theorem 4.7 follows from Lemma 4.7 and Observation 4.6.

Note that this instance can be generalized to arbitrary larger instances with the same lower bound as well. For example, we can double the size of the above instance above as follows. We create a copy of the above graph with nodes $1', \ldots 7'$. We 'connect' these two graphs by saying that node 7 and node $1'$ are the same node. Hence, node 7 will also be adjacent to nodes $2'$ and $3'$.

By adding copies of the graph as described above we can create larger instances with the same lower bound.

4.5 BAD INSTANCES FOR A NATURAL GREEDY ALGORITHM

An informal description of a natural greedy algorithm for problem PQ (and its special cases) is as follows: repeatedly select, among all possible quads, a quad with lowest cost, and remove the vectors in the selected quad from the instance; stop when no more vectors remain.

Below we present instances of problem PQ, as well as of its special case PQ(#1 $= 2$, distinct, connected) showing that the worst-case ratio of this greedy algorithm is worse than the worst-case ratio of algorithm A.

4.5.1 *An instance of PQ*

Consider the following instance I of PQ consisting of the following 12 vectors, $v_1, \ldots, v_{12}$:

$$\begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}, \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}, \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}, \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix},$$

$$\begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}, \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}, \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}.$$

Clearly, an optimum solution consists of the quads $\{v_1, v_4, v_5, v_6\}$, $\{v_2, v_7, v_8, v_9\}$ and $\{v_3, v_{10}, v_{11}, v_{12}\}$, with a total cost of $OPT(I) = 3$. The greedy algorithm however, may first select quad $\{v_1, v_2, v_3, v_4\}$ with cost 1. Then the cheapest possible quad is one of cost 2 and the greedy algorithm may select $\{v_7, v_8, v_{11}, v_{12}\}$. The remaining vectors form a quad of cost 3. Thus, the greedy algorithm finds a solution of cost $6 = 2 \cdot \mathrm{OPT}(I)$.

4.5.2 *An instance of PQ*(#1 = 2, *distinct, connected*)

Consider the following instance I of PQ(#1 = 2, distinct, connected) consisting of 8 vectors represented in a graph shown in Figure 25 (recall that a vector in PQ(#1 = 2, distinct, connected) corresponds to an edge in a graph).

An optimal solution for this instance has cost 10, with the two quads $\{(1,2),(2,5),(3,5),(3,4)\}$ and $\{(6,7),(5,7),(5,8),(8,9)\}$, each having cost 5.

Since the instance features no quad with cost 4, the greedy algorithm may first select the following quad with cost 5: $\{(2,5),(3,5),(5,7),(5,8)\}$. Next, what remains is a quad of cost 8: $\{(1,2),(3,4),(6,7),(8,9)\}$.

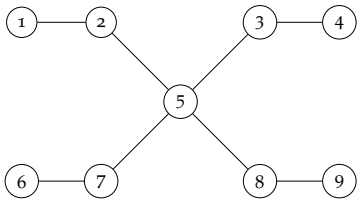


Figure 25: An instance of PQ(#1 = 2, distinct, connected)

Hence, the worst-case ratio of the greedy algorithm is at least 13/10, which is larger than the 5/4 approximation guarantee for algorithm A.

4.6 CONCLUSION

We have studied the worst-case behavior of a matching-based algorithm for partitioning a given set of vectors into quadruples. Informally, by running a matching algorithm once, we find pairs, and by running it one more time, we match the pairs into quadruples. Under the specific cost-structure studied here, we have shown the precise worst-case behavior of this method.

It is a natural question to study an extension where we form clusters consisting of 2^s vectors for some given integer $s \geqslant 2$. Indeed, if we form groups of size 2^s by running s rounds of matching, the worst-case ratio is easily seen to be bounded by 2^{s-1}. To explain this, let M^1 be the minimum-cost matching of the first round. Then $A(I) \leqslant \text{cost}(M)$ and $OPT(I) \geqslant \text{cost}(M)/2^{s-1}$ as the cost of the optimum (viewed as being constructed in s rounds) is at least $\text{cost}(M^1)$ after the first round and could then halve in each further round. Moreover, since we have shown that the cost of the algorithm after two rounds is at most $3/2$ times the optimal cost after two rounds, we get a ratio of $3/2 \times 2^{s-2} = 3 \times 2^{s-3}$. We leave the question of finding the worst-case ratio for arbitrary s as an open problem.

LIST OF FIGURES

Figure 1 Robust Balanced: an example 8
Figure 2 Robust Balanced: q-uniform set system 21
Figure 3 Robust Balanced: spanning tree 24
Figure 4 Robust Balanced: s, t-cut 26
Figure 5 Robust Balanced: connecting path 29
Figure 6 TPC: an example 54
Figure 7 TPC: Conflict Graph G_{C_1} 55
Figure 8 TPC: Conflict Graph G_{F_7} 55
Figure 9 TPC: Red-Blue Transportation 56
Figure 10 TPC: maximum independent set 61
Figure 11 TPC: conflict graph is a matching 66
Figure 12 TPC: conflict graph is a simple path 69
Figure 13 TPC: conflict graph is a split graph 72
Figure 14 TPC: conflict graph is complete bipartite 75
Figure 15 PQ: an example 85
Figure 16 PQ: auxiliary graph for PQ 100
Figure 17 PQ: binary expansions of a quad 105
Figure 18 PQ: $PQ(\#1 = 2)$, $cost(Q) = 2$ 109
Figure 19 PQ: $PQ(\#1 = 2)$, $cost(Q) = 3$ 109
Figure 20 PQ: $PQ(\#1 = 2)$, $cost(Q) = 4$ 111
Figure 21 PQ: $PQ(\#1 = 2)$, $cost(Q) = 5$ 113
Figure 22 PQ: $PQ(\#1 = 2, \text{distinct, connected})$, $cost(Q) = 4$. 120
Figure 23 PQ: worst-case $PQ(\#1 = 2, \text{connected})$ 123
Figure 24 PQ: worst-case $PQ(\#1 = 2, \text{distinct, connected})$ . . 124
Figure 25 PQ: bad instance greedy 127

LIST OF TABLES

Table 1 Robust Balanced: Class 1, Uniform 47
Table 2 Robust Balanced: Class 1, Normal mean 50 47
Table 3 Robust Balanced: Class 1, Normal mean 500 . . . 48
Table 4 Robust Balanced: Class 2, Uniform 48
Table 5 Robust Balanced: Class 2, Normal mean 50 49
Table 6 Robust Balanced: Class 2, Normal mean 500 . . . 49
Table 7 Robust Balanced: Class 3, Uniform 50
Table 8 Robust Balanced: Class 3, Normal 50
Table 9 PQ: overview of results 96
Table 10 PQ: PQ(#1 $= 2$, distinct, connected) cost(Q) $= 4$. 120

BIBLIOGRAPHY

Ahuja, R. (1997). The balanced linear programming problem. *European Journal of Operational Research*, 101(1):29–38.

Ahuja, R. K., Magnanti, T. L., and Orlin, J. B. (1993). *Network flows: theory, algorithms, and applications*. Prentice Hall.

Aissi, H., Bazgan, C., and Vanderpooten, D. (2005). Complexity of the min–max and min–max regret assignment problems. *Operations Research Letters*, 33(6):634–640.

Aissi, H., Bazgan, C., and Vanderpooten, D. (2009). Min–max and min–max regret versions of combinatorial optimization problems: A survey. *European Journal of Operational Research*, 197(2):427 – 438.

Bar-Noy, A. and Kortsarz, G. (1998). Minimum color sum of bipartite graphs. *Journal of Algorithms*, 28(2):339 – 365.

Barát, J. and Gerbner, D. (2014). Edge-decomposition of graphs into copies of a tree with four edges. *The Electronic Journal of Combinatorics*, 21(1):P1–55.

Ben-Tal, A., Boyd, S., and Nemirovski, A. (2006). Extending scope of robust optimization: Comprehensive robust counterparts of uncertain problems. *Mathematical Programming*, 107(1-2):63–89.

Ben-Tal, A., El Ghaoui, L., and Nemirovski, A. (2009). *Robust optimization*. Princeton University Press.

Ben-Tal, A., Golany, B., Nemirovski, A., and Vial, J.-P. (2005). Retailer-supplier flexible commitments contracts: A robust optimization approach. *Manufacturing & Service Operations Management*, 7(3):248–271.

Ben-Tal, A. and Nemirovski, A. (1998). Robust convex optimization. *Mathematics of Operations Research*, 23(4):769–805.

Ben-Tal, A. and Nemirovski, A. (1999). Robust solutions of uncertain linear programs. *Operations Research Letters*, 25(1):1–13.

Ben-Tal, A. and Nemirovski, A. (2000). Robust solutions of linear programming problems contaminated with uncertain data. *Mathematical Programming*, 88(3):411–424.

Bender, M., Thielen, C., and Westphal, S. (2015). Packing items into several bins facilitates approximating the separable assignment problem. *Information Processing Letters*, 115(6):570 – 575.

Berge, C. (1958). Sur le couplage maximum d'un graphe. *Comptes Rendus de l'Académie des Sciences*, 247:258–259.

Bertsimas, D., Brown, D. B., and Caramanis, C. (2011). Theory and applications of robust optimization. *SIAM Review*, 53(3):464 – 501.

Bertsimas, D. and Sim, M. (2003). Robust discrete optimization and network flows. *Mathematical Programming*, 98(1):49 – 71.

Camerini, P., Maffioli, F., Martello, S., and Toth, P. (1986). Most and least uniform spanning trees. *Discrete Applied Mathematics*, 15(2-3):181 – 197.

Cao, B. (1992). Transportation problem with nonlinear side constraints a branch and bound approach. *Zeitschrift für Operations Research*, 36(2):185–197.

Cao, B. and Uebe, G. (1995). Solving transportation problems with nonlinear side constraints with tabu search. *Computers & Operations Research*, 22(6):593 – 603.

Cappanera, P. and Scutellà, M. (2005). Balanced paths in acyclic networks: Tractable cases and related approaches. *Networks*, 45(2):104–111.

Chen, C., Zheng, L., Srinivasan, V., Thomo, A., Wu, K., and Sukow, A. (2016). Conflict-aware weighted bipartite b-matching and its application to e-commerce. *IEEE Transactions on Knowledge and Data Engineering*, 28(6):1475–1488.

Darmann, A., Pferschy, U., Schauer, J., and Woeginger, G. J. (2011). Paths, trees and matchings under disjunctive constraints. *Discrete Applied Mathematics*, 159(16):1726 – 1735. 8th Cologne/Twente Workshop on Graphs and Combinatorial Optimization (CTW 2009).

Deineko, V. and Woeginger, G. (2006). On the robust assignment problem under a fixed number of cost scenarios. *Operations Research Letters*, 34:175 – 179.

Dokka, T., Bougeret, M., Boudet, V., Giroudeau, R., and Spieksma, F. C. (2013). Approximation algorithms for the wafer to wafer integration problem. In *Proceedings of the 10th Workshop Approximation and Online Algorithms (WAOA 2012)*, pages 286–297. Springer Berlin Heidelberg.

Dokka, T., Crama, Y., and Spieksma, F. (2014). Multi-dimensional vector assignment problems. *Discrete Optimization*, 14:111 – 125.

Dong, F., Yan, W., and Zhang, F. (2013). On the number of perfect matchings of line graphs. *Discrete Applied Mathematics*, 6(161):794–801.

Dunning, I., Huchette, J., and Lubin, M. (2017). Jump: A modeling language for mathematical optimization. *SIAM Review*, 59(2):295–320.

Ficker, A. M. C., Spieksma, F. C. R., and Woeginger, G. J. (2017). Balanced optimization with vector costs. In *Proceedings of the 14th Workshop Approximation and Online Algorithms (WAOA 2016)*, pages 92–102. Springer International Publishing.

Ficker, A. M. C., Spieksma, F. C. R., and Woeginger, G. J. (2018). Robust balanced optimization. *EURO Journal on Computational Optimization*.

Figueroa, A., Goldstein, A., Jiang, T., Kurowski, M., Lingas, A., and Persson, M. (2005). Approximate clustering of fingerprint vectors

with missing values. In *Proceedings of the 2005 Australasian symposium on Theory of computing-Volume 41*, pages 57–60. Australian Computer Society, Inc.

Fleischer, L., Goemans, M. X., Mirrokni, V. S., and Sviridenko, M. (2011). Tight approximation algorithms for maximum separable assignment problems. *Mathematics of Operations Research*, 36(3):416–431.

Gabrel, V., Murat, C., and Thiele, A. (2014). Recent advances in robust optimization: An overview. *European Journal of Operational Research*, 235(3):471 – 483.

Galil, Z. and Schieber, B. (1988). On finding most uniform spanning trees. *Discrete Applied Mathematics*, 20(2):173 – 175.

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

Goossens, D., Polyakovskiy, S., Spieksma, F. C. R., and Woeginger, G. J. (2012). Between a rock and a hard place: the two-to-one assignment problem. *Mathematical Methods of Operations Research*, 76(2):223–237.

Goossens, D. R. and Spieksma, F. C. R. (2009). The transportation problem with exclusionary side constraints. *4OR*, 7(1):51–60.

Gorissen, B. L., Yanıkoğlu, İ., and den Hertog, D. (2015). A practical guide to robust optimization. *OMEGA*, 53:124–137.

Håstad, J. (1996). Clique is hard to approximate within $n^{1-\epsilon}$. In *Proceedings of 37th Conference on Foundations of Computer Science*, pages 627–636.

Hochbaum, D. S. and Levin, A. (2010). Covering the edges of bipartite graphs using $K_{2,2}$ graphs. *Theoretical Computer Science*, 411(1):1 – 9.

Holyer, I. (1981). The NP-completeness of some edge-partition problems. *SIAM Journal on Computing*, 10(4):713–717.

Jünger, M., Reinelt, G., and Pulleyblank, W. R. (1985). On partitioning the edges of graphs into connected subgraphs. *Journal of Graph Theory*, 9(4):539–549.

Kalra, T., Mathew, R., Pal, S. P., and Pandey, V. (2017). Maximum weighted independent sets with a budget. In *Conference on Algorithms and Discrete Applied Mathematics (CALDAM 2017)*, pages 254–266. Springer International Publishing.

Kamura, Y. and Nakamori, M. (2014). Modified balanced assignment problem in vector case: System construction problem. In *Proceedings of the International Conference on Computational Science and Computational Intelligence (CSCI 2014)*, volume 2, pages 52–56. IEEE Computer Society.

Katoh, N. and Iwano, K. (1994). Efficient algorithms for minimum range cut problems. *Networks*, 24(7):395–407.

Khot, S. (2001). Improved inapproximability results for maxclique, chromatic number and approximate graph coloring. In *Proceedings of 42nd Annual Symposium on Foundations of Computer Science*, pages 600–609.

Kinable, J., Smeulders, B., Delcour, E., and Spieksma, F. (2017). Exact algorithms for the equitable traveling salesman problem. *European Journal of Operational Research*, 261(2):475–485.

Koster, A. M., Kutschka, M., and Raack, C. (2013). Robust network design: Formulations, valid inequalities, and computations. *Networks*, 61(2):128–149.

Kouvelis, P. and Yu, G. (1997). *Robust discrete optimization and its applications*. Kluwer Academic Publishers, Norwell, Massachusets.

Larusic, J. and Punnen, A. (2011). The balanced traveling salesman problem. *Computers & Operations Research*, 38(5):868–875.

Lee, C., Lee, K., Park, K., and Park, S. (2012). Branch-and-price-and-cut approach to the robust network design problem without flow bifurcations. *Operations Research*, 60(3):604–610.

Martello, S., Pulleyblank, W., Toth, P., and De Werra, D. (1984). Balanced optimization problems. *Operations Research Letters*, 3(5):275–278.

Marx, D. (2005). A short proof of the NP-completeness of minimum sum interval coloring. *Operations Research Letters*, 33(4):382 – 384.

Onn, S. and Schulman, L. J. (2001). The vector partition problem for convex objective functions. *Mathematics of Operations Research*, 26(3):583–590.

Pferschy, U. and Schauer, J. (2013). The maximum flow problem with disjunctive constraints. *Journal of Combinatorial Optimization*, 26(1):109–119.

Poss, M. (2014). Robust combinatorial optimization with variable cost uncertainty. *European Journal of Operational Research*, 237:836–845.

Punnen, A. and Nair, K. (1999). Constrained balanced optimization problems. *Computers & Mathematics with Applications*, 37(9):157–163.

Ratner, D. and Warmuth, M. (1990). The $(n^2 - 1)$-puzzle and related relocation problems. *Journal of Symbolic Computation*, 10(2):111 – 137.

Reda, S., Smith, G., and Smith, L. (2009). Maximizing the functional yield of wafer-to-wafer 3-D integration. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 17(9):1357–1362.

Salavatipour, M. (2000). On sum coloring of graphs. Master's thesis, Department of Computer Science, University of Toronto.

Sun, M. (1998). A tabu search heuristic procedure for solving the transportation problem with exclusionary side constraints. *Journal of Heuristics*, 3(4):305–326.

Sun, M. (2002). The transportation problem with exclusionary side constraints and two branch-and-bound algorithms. *European Journal of Operational Research*, 140(3):629 – 647.

Syarif, A. and Gen, M. (2003). Solving exclusionary side constrained transportation problem by using a hybrid spanning tree-based genetic algorithm. *Journal of Intelligent Manufacturing*, 14(3-4):389–399.

Szkaliczki, T. (1999). Routing with minimum wire length in the dogleg-free manhattan model is $\mathcal{NP}$-complete. *SIAM Journal on Computing*, 29(1):274–287.

Thomassen, C. (2008). Edge-decompositions of highly connected graphs into paths. In *Abhandlungen aus dem Mathematischen Seminar der Universität Hamburg*, volume 78, pages 17 – 26. Springer.

Turner, L. (2012). Variants of shortest path problems. *Algorithmic Operations Research*, 6(2):91–104.

Vancroonenburg, W., Della Croce, F., Goossens, D., and Spieksma, F. C. R. (2014). The red-blue transportation problem. *European Journal of Operational Research*, 237(3):814 – 823.

Vazirani, V. (2001). *Approximation Algorithms*. Springer-Verlag, Inc., New York, USA.

Wiesemann, W., Kuhn, D., and Sim, M. (2014). Distributionally robust convex optimization. *Operations Research*, 62(6):1358–1376.

Williamson, D. P. and Shmoys, D. B. (2011). *The Design of Approximation Algorithms*. Cambridge University Press, New York, USA, 1st edition.

DOCTORAL DISSERTATIONS FROM THE FACULTY OF ECONOMICS AND BUSINESS

Doctoral dissertations from the Faculty of Economics and Business, see: `http://www.kuleuven.ac.be/doctoraatsverdediging/archief.htm`.