EURO J Comput Optim (2018) 6:239–266
https://doi.org/10.1007/s13675-018-0093-y

ORIGINAL PAPER

Robust balanced optimization

Annette M. C. Ficker[1] · **Frits C. R. Spieksma**[2] · **Gerhard J. Woeginger**[3]

Received: 13 July 2017 / Accepted: 1 February 2018 / Published online: 12 February 2018

Abstract An instance of a balanced optimization problem with vector costs consists of a ground set X, a cost-vector for every element of X, and a system of feasible subsets over X. The goal is to find a feasible subset that minimizes the so-called imbalance of values in every coordinate of the underlying vector costs. Balanced optimization problems with vector costs are equivalent to the robust optimization version of balanced optimization problems under the min-max criterion. We regard these problems as a family of optimization problems; one particular member of this family is the known balanced assignment problem. We investigate 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 by performing computational experiments for this problem.

A preliminary version of this work has appeared in the Proceedings of the 14th International Workshop on Approximation and Online Algorithms (WAOA 2016), 92–102.

✉ Annette M. C. Ficker
annette.ficker@kuleuven.be

Frits C. R. Spieksma
f.c.r.spieksma@tue.nl

Gerhard J. Woeginger
woeginger@cs.rwth-aachen.de

1 Faculty of Economics and Buisness, KU Leuven, 3000 Leuven, Belgium

2 Department of Mathematics and Computer Science, Eindhoven University of Technology, 5600 MB Eindhoven, The Netherlands

3 Lehrstuhl für Informatik 1, RWTH Aachen, 52056 Aachen, Germany

Keywords Balanced optimization · Assignment problem · Computational complexity · Approximation

Mathematics Subject Classification 90C27

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 (Martello et al. 1984), we are given an $n \times n$ matrix C with real entries $c(i, j)$ for $1 \leq i, j \leq n$. 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

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

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

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

The imbalance in the k-th coordinate of assignment A (with $1 \leq k \leq d$) 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 $\Delta_{\max}(A)$. Note that for $d = 1$ we recover the traditional balanced assignment problem.

Balanced optimization problems with vector costs are closely connected to robust optimization (see Sect. 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 scenario's, we receive a set of scenario's 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, in case of a discrete set of scenario's 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 $1 \leq i, j \leq n$. 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. For instance, Kamura and Nakamori (2014) sketch an industrial problem in the manufacturing of glass lenses that gives rise to a (specially structured) balanced assignment problem with vector costs; see Sect. 5 for more details on this.

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-coefficients. Another option is to model uncertainty by using settings where a discrete number of scenario's 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 (MST $(|V|, |E|) + |V| \log |V|)$, where 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 $x \geq 0$) and costs associated with each real variable x_i. The goal is to minimize the difference between the largest nonzero cost $c_i x_i$ and the smallest nonzero 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.

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 Sect. 2.
- Every problem in our framework (i) is solvable in polynomial time if the dimension d is fixed (see Sect. 3.1), and (ii) allows a polynomial time 2-approximation algorithm (see Sect. 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 Sect. 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 Sect. 4.4). 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 $\frac{4}{3}$ implies $P = NP$; see Sect. 5.
- We perform extensive computational experiments, investigating the computational behavior of an integer programming formulation and 2-approximation algorithms on different classes of instances of the robust balanced assignment problem in Sect. 6.

2 The framework

Throughout this paper, 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, returns 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 the so-called *robust balanced* versions of the problems in the framework. For this, we generalize the terminology introduced in Sect. 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 $(1 \le k \le d)$ 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

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

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.

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 (Sect. 3.1). The second and third algorithms both yield a 2-approximation in polynomial time (Sect. 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 $n := |X|$.

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 in 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 values. 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 sufficient to try all possibilities for the lowest value of each coordinate in an optimum solution, as is argued in Theorem 1.

Algorithm 1

```
1: Sol := $\infty$
2: **for** each $x, y \in X$ **let** $\Delta := \Delta_{\max}(\{x, y\})$ **do**
3:    **for** each $x_1 \in X$ **let** $\min_1 := c_1(x_1)$ **do**
4:                $\vdots$
5:       **for** each $x_d \in X$ **let** $\min_d := c_d(x_d)$ **do**
6:          $Y := X$
7:          **for** each $z \in X$ **do**
8:             **if** $\exists k$ such that $c_k(z) < \min_k$ or $c_k(z) > \min_k + \Delta$ **then**
9:                $Y := Y \setminus \{z\}$
10:         **if** $Y$ contains a feasible solution **then**
11:            Sol := $\min\{\text{Sol}, \Delta\}$
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 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 $\Delta_{\max}(\{x, y\}) = OPT$. 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 $d = 1$.

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

```
1: for each pair $x_1, x_2$ in ground set $X$ do
2:     $\Delta := \Delta_{\max}(\{x_1, x_2\})$
3:     $Y := X$
4:     for each $x \in X$ do
5:         if $\max_k |c_k(x_1) - c_k(x)| > \Delta$ or $\max_k |c_k(x_2) - c_k(x)| > \Delta$ then
6:             $Y := Y \setminus \{x\}$
7:     if $Y$ contains a feasible solution then
8:         $\mathrm{Sol}(x_1, x_2) := \Delta_{\max}(Y)$
9: $\mathrm{Sol} := \min_{x_1,x_2} \mathrm{Sol}(x_1, x_2)$
10: Output Sol.
```

Theorem 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, $\Delta_{\max}(\{x_1, x_2\}) = OPT$.

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

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

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.

Algorithm 3

```
1: Sol := $\infty$
2: **for** each $x \in X$ **do**
3:     set $S_x := \emptyset$
4:     **for** each $y \in X$ **do**
5:         $S_x = S_x \cup \{\Delta_{\max}(\{x, y\})\}$
6:     Sort $S_x$ in nondecreasing order
7:     **while** Binary Search on $S_x$ **do**
8:         select $\Delta \in S_x$
9:         $Y := X$
10:        **for** each $y \in X$ **do**
11:            **if** $\Delta_{\max}(\{x, y\}) > \Delta$ **then**
12:                $Y := Y \backslash \{y\}$
13:        **if** $Y$ contains a feasible solution **then**
14:            **if** $\Delta <$ Sol **then**
15:                Sol := $\Delta$
16:            select $\Delta' \in S_x$ with $\Delta' < \Delta$ according to the Binary Search
17:        **else** select $\Delta' \in S_x$ with $\Delta' > \Delta$ according to the Binary Search
18: Output Sol.
```

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.

4 The complexity of robust balanced optimization problems

Many balanced optimization problems with scalar costs are known to be solvable in polynomial time (see the discussion in Sect. 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 claim 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 give explicit proofs for the robust balanced q-uniform set system problem (Sect. 4.1) and for the robust balanced assignment problem (Sect. 4.2). For the other problems, we state the results in Sect. 4.3 and we refer to Ficker et al. (2018) for the proofs. 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 (Sect. 4.4).

We use the following problem, well known to be NP-complete, in the reductions present in this paper.

Problem: INDEPENDENT SET (IS)
Instance: A graph $G = (V, E)$ with vertex set $V = \{v_1, \dots, v_n\}$ and edge set $E = \{e_1, \dots, e_m\}$; 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?

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.

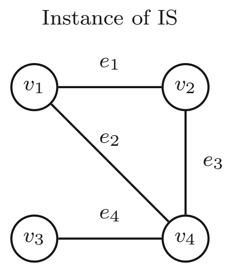
Theorem 3 *The robust balanced q-uniform set system problem is NP-hard.*

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, $1 \le k \le m$. Let us now define cost-vector $\mathbf{c}(v_j) = (c_1(v_j), c_2(v_j), \dots, c_d(v_j))$ corresponding to each vertex $v_j \in V$. For each $v_j \in V$ and $k \in \{1, \dots, m\}$:

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

This specifies the corresponding instance of the robust balanced q-uniform set system, see Fig. 1 for an illustration. We claim that there exists a feasible subset $Y \subseteq X$ with $|Y| = z$ and $\Delta_{\max}(Y) \le 1$ 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 $\Delta_{\max}(Y) \le 1$. 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



Corresponding instance q-Uniform Set System

$$\mathbf{c}(v_1) = (1, 1, 0, 0)$$

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

$$\mathbf{c}(v_3) = (0, 0, 0, 1)$$

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

Fig. 1 Illustration of the construction of an instance of the robust balanced q-uniform set system

of e_k is -1, and the cost-vector of the target vertex of e_k is $+1$. Hence $\Delta_{\max}(Y) \geq 2$. 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 $\Delta_{\max}(I) \leq 1$. □

Corollary 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 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. □

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 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 $(1 \leq i \leq n)$. 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 $\mathbf{c}(a, b)$, $1 \leq a, b \leq n$; 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 $b \leq z$, 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 $b \geq z + 1$, 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 $b \leq z$. 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 *The robust balanced assignment problem does not allow a polynomial time approximation algorithm with worst-ase guarantee strictly better than 2 (unless P = NP).*

4.3 Other robust balanced optimization problems

We now state our results for the remaining problems.
Recall that the robust balanced spanning tree can be described as follows: given a graph with a cost-vector for each edge, find a spanning tree in this graph minimizing the imbalance.

Theorem 5 *The robust balanced spanning tree problem is NP-hard and does not allow a polynomial time approximation algorithm with worst-ase guarantee strictly better than* 2 *(unless P = NP).*

Proof See Ficker et al. (2018). □

Another problem in our framework is the robust balanced s, t-cut problem. Given a graph with a cost-vector for each edge, and two nodes s and t, the problem is to find a cut in this graph, separating s and t, which minimizes the imbalance.

Theorem 6 *The robust balanced s, t-cut problem is NP-hard and does not allow a polynomial time approximation algorithm with worst-ase guarantee strictly better than* 2 *(unless P = NP).*

Proof See Ficker et al. (2018). □

In the robust balanced connecting path problem, a graph is given with a cost-vector for each edge, and two nodes s and t. The problem is to find a path connecting s and t which minimizes the imbalance.

Theorem 7 *The robust balanced connecting path problem is NP-hard and does not allow a polynomial time approximation algorithm with worst-ase guarantee strictly better than* 2 *(unless P = NP).*

Proof See Ficker et al. (2018). □

The last problem we consider in this section is the robust balanced Horn-SAT problem. In this problem, we are given a set of literals X with vector costs and a set of clauses over these literals, where each clause contains at most 1 positive literal; the problem is to find a satisfying truth assignment which minimizes the imbalance.

Theorem 8 *The robust balanced Horn-SAT problem is NP-hard, and does not allow a polynomial time approximation algorithm with worst-case guarantee strictly better than* 2 *(unless P = NP).*

Proof See Ficker et al. (2018). □

4.4 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 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., $\bar{\bar{x}} = x$.)

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. □

5 A special case of the robust balanced assignment problem: 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 10 *The robust balanced assignment problem with sum costs is NP-hard.*

Proof We modify the hardness construction used in the proof of Theorem 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 $(1 \leq i \leq n)$. The other z rows $a := n + j$ and z columns $b := n + j$ in the matrix C $(1 \leq j \leq z)$ 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, $1 \leq a \leq n + z$. There are two cases:

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

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

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

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

The resulting matrix C can be represented as follows:

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

We now show that the existence of an IS with size z implies the existence of a solution with imbalance at most $\frac{3}{2}$, whereas the nonexistence 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 ($a \le n$), and any column $b \ge n+1$ (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 ($a \le n$), 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 $\frac{3}{2}$.

Suppose no size z-independent set exists. We claim that any assignment has imbalance ≥ 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 ≥ 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 ≥ 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 3 *The robust balanced assignment problem with sum costs does not allow a polynomial time approximation algorithm with worst-case guarantee strictly better than $\frac{4}{3}$ (unless P = NP).*

Remark 1 Corollary 3 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.

$$\left[C = \begin{pmatrix} (0,-1) & (1,0) & (-1,-2) \\ (0,0) & (1,1) & (-1,-1) \\ (1,0) & (2,1) & (0,-1) \end{pmatrix} \right]$$

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

There is a straightforward reduction from Numerical 3-Dimensional Matching, we refer to Ficker et al. (2018) for the proof.

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 Sect. 1).

6.1 MIP formulation

The following 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), $1 \leq i, j \leq n$, real variables u_k and ℓ_k denote the, respectively, largest and smallest value in dimension k of a selected entry ($1 \leq k \leq d$), and the real variable t captures the objective function value.

$$\text{minimize} \quad t \tag{2a}$$

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

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

$$c_k(i,j)\cdot x_{i,j} \le u_k \qquad i,j=1,\ldots,n;\ k=1,\ldots,d, \tag{2d}$$

$$\ell_k \le c_k(i,j)\cdot x_{i,j} + M(1-x_{i,j}) \qquad i,j=1,\ldots,n;\ k=1,\ldots,d, \tag{2e}$$

$$u_k - \ell_k \le t \qquad k=1,\ldots,d, \tag{2f}$$

$$x_{i,j}\in\{0,1\} \quad i,j=1,\ldots,n, \tag{2g}$$

$$u_k, \ell_k, t \ge 0 \quad k=1,\ldots,d. \tag{2h}$$

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.

6.2 Construction of datasets

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 scenario's, where the k-th component of each cost-vector corresponds to scenario k, $k=1,\ldots,d$. 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 scenario's, where the first scenario can be seen as the most "average" scenario, and the last scenario is the most "extreme" scenario; in between, the scenario's 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 scenario's 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 scenario's 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 ($k = 1, \dots, d$). For each $d \in \{2, 100, 300\}$, we use the following four collections of probability distributions.
  - $U\left(\left\lceil 50 - \frac{50k}{d} \right\rceil, \left\lfloor 50 + \frac{50k}{d} \right\rfloor\right)$,
  - $U\left(\left\lceil 500 - \frac{500k}{d} \right\rceil, \left\lfloor 500 + \frac{500k}{d} \right\rfloor\right)$,
  - $N(50; \frac{50k}{3d})$, 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 scenario's coming from different distributions leads to other instances in terms of solvability.

We have used R 3.1.1 to generate the data.

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 @2.70 GHz and 16 GB 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 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 Sect. 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 $\text{Sol}(x_1, x_2) := \Delta_{\max}(Y)$, we set $\text{Sol}(x_1, x_2)$ 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.

6.4 Results

Hereunder we present five tables to present our computational results. Tables 1 and 2 contain the results of instances belonging to Class 1, Tables 3 and 4 show the results of instances belonging to class 2, and Table 5 shows the results of instances belonging to Class 3. Each table consists of several multicolumns. One multicolumn corresponds to the instances generated from one probability distribution and contains the results for the MIP, Algorithms 2 and 3 . For each of these we show: the computing time in seconds (time), the solution value that was found (Sol), the coordinate in which this solution value is attained (arg_k), and for Algorithms 2 and 3 we show the gap compared to the solution found by the MIP.

Recall that each entry is the average over 10 instances. Also 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 s 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 s to solve. For $n = 20$, instances with $d = 100$ are often not solved to optimality

Table 1 Results Class 1, distributions with mean 50

	U(0,100)			N(50,1)			N(50,10)		
	MIP	Alg2	Alg3	MIP	Alg2	Alg3	MIP	Alg2	Alg3
n10 d2									
Time	0.08	0.56	0.59	0.10	0.40	0.56	0.09	0.45	0.54
Sol	38.40	38.70	39.40	1.03	1.04	1.10	11.03	11.05	11.61
$\arg_k$	1.65	1.55	1.45	1.60	1.40	1.70	1.70	1.70	1.60
Gap (%)		0.78	2.60		0.26	6.48		0.12	5.22
n10 d100									
Time	6.58	11.12	1.03	8.98	9.66	1.30	8.97	11.15	1.31
Sol	94.10	95.30	96.70	3.93	4.05	4.32	39.69	41.61	43.28
$\arg_k$	41.43	57.67	46.15	48.60	60.40	55.30	39.00	51.10	54.00
Gap (%)		1.28	2.76		3.09	9.87		4.83	9.03
n10 d300									
Time	49.12	24.19	1.93	73.34	26.72	2.65	59.81	24.77	2.66
Sol	97.40	98.00	98.80	4.48	4.68	4.90	44.90	46.33	48.42
$\arg_k$	212.98	157.12	148.72	173.40	102.80	89.60	183.50	172.40	151.70
Gap (%)		0.62	1.44		4.38	9.31		3.18	7.82
n20 d2									
Time	0.34	16.69	10.74	0.46	15.02	2.69	0.49	15.02	4.87
Sol	31.40	32.10	33.70	0.94	0.96	1.01	9.03	9.22	9.77
$\arg_k$	1.30	1.35	1.45	1.30	1.50	1.70	1.70	1.70	1.40
Gap (%)		2.23	7.32		2.09	7.64		2.12	8.29
n20 d100									
Time	(9)*	793.39	13.21	(2)*	876.10	21.44	(8)*	873.56	21.95
Sol	95.00	98.00	98.70	4.00	4.43	4.70	40.38	44.59	47.04
$\arg_k$	52.36	46.11	52.07	51.90	61.70	62.10	27.50	52.70	48.20
Gap (%)		3.16	3.89		10.61	17.36		10.44	16.49
n20 d300									
Time	(0)*	1623.0	26.53	(0)*	2088.4	50.85	(0)*	2091.7	51.49
Sol	98.30	99.00	99.80	4.67	5.05	5.17	46.80	50.17	51.62
$\arg_k$	142.55	143.78	153.02	180.10	174.10	178.10	147.60	129.90	158.70
Gap (%)		0.71	1.53		8.20	10.73		7.20	10.31

* Number of solved instances within 3600 s

by the MIP, and, except for instances of Class 2, instances with $n = 20$ and $d = 300$ cannot be solved within 3600 s.

Of course, the approximation algorithms fare better than the MIP in terms of computation times: each of them always finds a solution within 3600 s. 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 s, while Algorithm 2 may need more than 2000 s.

Table 2 Results Class 1, distributions with mean 500

	U(0,1000)			N(500,1)			N(500,10)		
	MIP	Alg2	Alg3	MIP	Alg2	Alg3	MIP	Alg2	Alg3
n10 d2									
Time	0.07	0.53	0.59	0.11	0.46	0.65	0.10	0.52	0.66
Sol	393.70	395.20	409.90	1.06	1.07	1.10	11.35	11.42	11.83
$\arg_k$	1.40	1.30	1.40	1.40	1.50	1.40	1.90	1.70	1.40
Gap (%)		0.38	4.11		1.04	3.97		0.59	4.16
n10 d100									
Time	12.01	11.07	1.39	8.70	11.09	1.63	8.32	11.38	1.85
Sol	936.20	947.60	959.30	3.96	4.13	4.32	39.69	41.42	43.50
$\arg_k$	66.15	52.23	61.70	60.00	57.30	40.10	51.60	58.30	43.30
Gap (%)		1.22	2.47		4.31	9.08		4.36	9.60
n10 d300									
Time	66.50	24.61	2.80	69.12	25.57	3.03	69.27	26.11	3.59
Sol	964.60	972.40	979.20	4.47	4.65	4.84	45.25	47.06	48.44
$\arg_k$	153.41	191.95	156.43	149.00	149.40	112.20	134.30	196.90	186.40
Gap (%)		0.81	1.51		3.98	8.16		4.00	7.05
n20 d2									
Time	0.40	11.81	7.73	0.51	11.25	3.09	0.46	10.23	5.16
Sol	312.30	315.10	343.30	0.91	0.93	0.99	8.72	8.84	9.20
$\arg_k$	1.45	1.55	1.50	1.50	1.40	1.40	1.40	1.40	1.30
Gap (%)		0.90	9.93		2.80	9.63		1.33	5.46
n20 d100									
Time	(2)*	847.48	21.74	(8)*	881.41	29.10	(9)*	884.40	29.71
Sol	942.00	967.90	977.40	4.02	4.47	4.71	40.10	44.68	46.74
$\arg_k$	46.03	56.60	42.80	69.20	35.90	58.70	46.80	47.00	56.10
Gap (%)		2.75	3.76		11.22	17.40		11.41	16.56
n20 d300									
Time	(0)*	1885.3	48.09	(0)*	2079.4	73.12	(0)*	2062.7	77.28
Sol	978.10	984.30	988.90	4.69	5.02	5.26	46.86	50.44	52.14
$\arg_k$	160.88	164.45	175.72	124.60	163.70	153.60	181.60	97.50	142.40
Gap (%)		0.63	1.10		7.06	12.00		7.64	11.28

* Number of solved instances within 3600 s

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 solution within 7% (12%) of the optimum value, while Algorithm 2 (Algorithm 3) finds solution within 14% (21%) of the optimum value for instances generated by the normal distribution. Here, it is also clear that, for both algorithms, performance degrades mildly with n,

Table 3 Results Class 2, distributions with mean 50

	U(0,100)			N(50,1)			N(50,10)		
	MIP	Alg2	Alg3	MIP	Alg2	Alg3	*MIP*	Alg2	Alg3
n10 d2									
Time	0.09	0.36	0.59	0.10	0.34	0.55	0.12	0.44	0.57
Sol	29.90	30.20	30.40	0.94	0.94	0.97	9.63	9.74	10.00
$\arg_k$	1.40	1.30	1.40	1.60	1.60	1.40	1.20	1.50	1.40
Gap (%)		1.00	1.67		0.05	3.12		1.12	3.82
n10 d100									
Time	2.39	5.29	0.95	5.89	2.82	1.34	6.63	2.77	1.31
Sol	98.50	98.50	98.50	4.60	4.60	4.65	46.24	46.25	46.62
$\arg_k$	97.75	98.25	98.43	98.90	98.80	98.90	98.50	98.60	98.50
Gap (%)		0.00	0.00		0.00	1.24		0.02	0.81
n10 d300									
Time	10.31	39.39	1.17	49.03	7.71	2.70	44.37	8.45	2.73
Sol	100.00	100.00	100.00	5.40	5.41	5.43	54.35	54.39	54.70
$\arg_k$	296.13	296.61	296.61	298.50	298.70	298.80	298.80	298.80	298.70
Gap (%)		0.00	0.00		0.08	0.61		0.06	0.64
n20 d2									
Time	0.29	12.67	7.39	0.44	9.38	7.13	0.41	7.90	7.03
Sol	22.80	23.00	24.10	0.71	0.72	0.76	7.10	7.18	7.77
$\arg_k$	1.45	1.55	1.75	1.30	1.30	1.50	1.70	1.60	1.60
Gap (%)		0.88	5.70		0.95	7.78		1.05	9.41
n20 d100									
Time	90.36	147.11	12.63	146.10	108.74	22.56	145.66	106.51	22.34
Sol	98.00	98.00	98.00	4.39	4.39	4.52	44.15	44.21	45.23
$\arg_k$	97.82	97.51	97.48	98.30	98.40	98.30	98.40	98.40	98.00
Gap (%)		0.00	0.00		0.14	3.04		0.14	2.45
n20 d300									
Time	84.71	2176.28	17.02	880.78	325.49	53.04	660.09	320.78	53.00
Sol	100.00	100.00	100.00	5.22	5.22	5.33	51.98	52.02	53.12
$\arg_k$	295.94	295.99	295.99	298.30	298.30	298.50	298.20	298.10	298.30
Gap (%)		0.00	0.00		0.09	2.10		0.09	2.19

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 3, 4): 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 s 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 (Table 3) solutions within

Table 4 Results Class 2, distributions with mean 500

	U(0,1000)			N(500,1)			N(500,10)		
	MIP	Alg2	Alg3	MIP	Alg2	Alg3	MIP	Alg2	Alg3
n10 d2									
Time	0.07	0.30	0.61	0.10	0.36	0.58	0.12	0.39	0.54
Sol	274.20	275.10	279.20	0.86	0.89	0.91	9.49	9.52	9.81
$\arg_k$	1.50	1.60	1.40	1.80	1.60	1.50	1.40	1.40	1.30
Gap (%)		0.33	1.82		2.78	5.72		0.29	3.35
n10 d100									
Time	4.57	2.15	1.32	4.54	2.48	1.35	6.45	2.94	1.30
Sol	977.10	977.10	978.20	4.55	4.55	4.58	46.68	46.77	47.38
$\arg_k$	99.05	98.75	98.75	98.50	98.50	98.40	98.80	98.80	98.80
Gap (%)		0.00	0.11		0.00	0.78		0.20	1.51
n10 d300									
Time	34.46	6.49	2.39	39.62	7.78	2.77	46.00	8.99	2.69
Sol	993.00	993.00	993.10	5.37	5.37	5.40	55.07	55.11	55.54
$\arg_k$	298.55	298.50	298.50	298.50	298.60	298.40	298.90	298.90	298.90
Gap (%)		0.00	0.01		0.01	0.48		0.07	0.84
n20 d2									
Time	0.40	10.06	7.67	0.45	9.41	6.93	0.47	8.82	7.09
Sol	233.60	236.60	253.90	0.71	0.74	0.76	6.95	7.09	7.27
$\arg_k$	1.40	1.30	1.50	1.40	1.40	1.50	1.50	1.70	1.40
Gap (%)		1.28	8.69		3.82	6.86		2.05	4.73
n20 d100									
Time	181.51	94.59	19.21	153.03	105.61	21.98	116.54	100.81	22.30
Sol	972.40	972.40	973.50	4.39	4.40	4.53	43.78	43.79	45.04
$\arg_k$	98.25	98.30	97.95	98.20	98.20	97.80	98.50	98.50	98.20
Gap (%)		0.00	0.11		0.02	3.04		0.02	2.87
n20 d300									
Time	855.57	289.24	37.43	731.45	309.28	52.92	630.81	331.84	52.69
Sol	990.90	991.00	991.40	5.19	5.20	5.31	52.29	52.32	53.13
$\arg_k$	297.83	298.15	297.80	298.30	298.30	297.90	298.40	298.40	298.10
gap (%)		0.01	0.05		0.03	2.20		0.06	1.60

1% of the optimum; for instances with mean 500 (see Table 4) 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 (Table 5).

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.

Table 5 Results Class 3

	U(0,100)			U(0,1000)			N(50,.)			N(500,.)		
	MIP	Alg2	Alg3	MIP	Alg2	Alg3	MIP	Alg2	Alg3	MIP	Alg2	Alg3
n10 d2												
Time	0.09	0.48	0.60	0.09	0.52	0.60	0.11	0.41	0.66	0.10	0.48	0.70
Sol	27.30	27.30	28.60	278.30	279.80	289.20	12.72	12.87	13.62	138.40	142.49	144.90
$\arg_k$	1.50	1.65	1.60	1.70	1.70	1.50	1.30	1.30	1.30	1.60	1.60	1.70
Gap (%)		0.0	4.76		0.54	3.92		1.17	7.09		2.96	4.69
n10 d100												
Time	5.76	7.99	1.08	6.61	7.37	1.35	6.95	8.48	1.98	6.54	8.70	1.90
Sol	76.20	78.50	80.40	761.80	774.20	803.20	47.84	49.20	52.93	477.34	496.99	518.87
$\arg_k$	86.68	91.95	92.62	87.80	88.60	94.20	86.60	91.90	79.20	83.80	83.90	82.30
Gap (%)		3.02	5.51		1.63	5.43		2.84	10.63		4.12	8.70
n10 d300												
Time	40.18	17.04	1.94	52.17	17.93	2.61	56.70	23.29	4.41	62.77	21.77	3.72
Sol	83.40	85.00	87.90	837.60	848.80	872.40	57.39	59.69	63.26	565.64	585.05	616.51
$\arg_k$	282.71	281.28	286.37	278.95	281.80	287.80	271.50	282.70	265.50	264.50	266.00	270.60
Gap (%)		1.92	5.40		1.34	4.15		4.00	10.22		3.43	8.99
n20 d2												
Time	0.35	16.48	7.11	0.40	12.20	7.46	0.43	10.96	9.50	0.47	9.52	9.25
Sol	22.20	22.30	24.00	222.00	223.70	232.80	10.88	11.01	11.97	104.06	105.67	110.60
$\arg_k$	1.60	1.55	1.55	1.50	1.55	1.55	1.60	1.60	1.70	1.80	1.40	1.50
Gap (%)		0.48	8.11		0.77	4.86		1.13	10.00		1.55	6.28

Table 5 continued

	U(0,100)			U(0,1000)			N(50,.)			N(500,.)		
	MIP	Alg2	Alg3	MIP	Alg2	Alg3	*MIP*	Alg2	Alg3	MIP	Alg2	Alg3
n20 d100												
Time	1155.9	571.98	14.01	906.52	599.53	21.46	741.16	731.24	30.97	824.02	737.70	29.97
Sol	76.10	80.90	85.00	758.30	801.30	835.30	47.50	53.30	57.27	475.08	536.97	570.32
$\arg_k$	90.37	89.63	94.17	87.45	90.60	92.00	82.80	79.60	90.10	88.10	87.40	89.50
Gap (%)		6.31	11.70		5.67	10.15		12.21	20.58		13.03	20.05
n20 d300												
Time	(0)*	1347.4	29.57	(0)*	1521.4	48.55	(1)*	1895.6	70.99	(0)*	1766.8	71.32
Sol	85.40	88.20	90.60	850.50	881.00	903.80	58.37	64.09	67.46	588.40	641.77	683.70
$\arg_k$	282.12	287.64	289.94	279.48	285.05	287.35	275.20	267.50	267.40	276.10	269.90	272.40
Gap (%)		3.28	6.09		3.59	6.27		9.81	15.99		9.07	16.2

* Number of solved instances within 3600 s

- The approximation algorithms offer a trade-off in terms of quality of solution found, and running time needed; in particular, Algorithm 3 finds quickly 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.

7 Conclusion

We introduce the notion of 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.

Acknowledgements We thank Marc Goerigk for interesting discussions on the relation between balanced optimization problems with vector costs and robust optimization. We are also indebted to the reviewers whose remarks led to a significant speedup of some of the algorithms in this work. This research has been supported by the Netherlands Organisation for Scientific Research (NWO) under Grant 639.033.403, by BSIK Grant 03018 (BRICKS: Basic Research in Informatics for Creating the Knowledge Society), and by the Interuniversity Attraction Poles Programme initiated by the Belgian Science Policy Office.

References

Ahuja R (1997) The balanced linear programming problem. Eur J Oper Res 101(1):29–38

Aissi H, Bazgan C, Vanderpooten D (2005) Complexity of the min-max and min-max regret assignment problems. Oper Res Lett 33(6):634–640

Aissi H, Bazgan C, Vanderpooten D (2009) Min-max and min-max regret versions of combinatorial optimization problems: a survey. Eur J Oper Res 197(2):427–438

Ben-Tal A, Nemirovski A (1998) Robust convex optimization. Math Oper Res 23(4):769–805

Ben-Tal A, Nemirovski A (1999) Robust solutions of uncertain linear programs. Oper Res Lett 25(1):1–13

Ben-Tal A, Nemirovski A (2000) Robust solutions of linear programming problems contaminated with uncertain data. Math Program 88(3):411–424

Ben-Tal A, Golany B, Nemirovski A, Vial J-P (2005) Retailer-supplier flexible commitments contracts: a robust optimization approach. Manuf Serv Oper Manag 7(3):248–271

Ben-Tal A, Boyd S, Nemirovski A (2006) Extending scope of robust optimization: comprehensive robust counterparts of uncertain problems. Math Program 107(1–2):63–89

Ben-Tal A, El Ghaoui L, Nemirovski A (2009) Robust optimization. Princeton University Press, Princeton

Bertsimas D, Sim M (2003) Robust discrete optimization and network flows. Math Program 98(1):49–71

Bertsimas D, Brown DB, Caramanis C (2011) Theory and applications of robust optimization. SIAM Rev 53(3):464–501

Camerini P, Maffioli F, Martello S, Toth P (1986) Most and least uniform spanning trees. Discrete Appl Math 15(2–3):181–197

Cappanera P, Scutellà M (2005) Balanced paths in acyclic networks: tractable cases and related approaches. Networks 45(2):104–111

Deineko V, Woeginger G (2006) On the robust assignment problem under a fixed number of cost scenarios. Oper Res Lett 34:175–179

Dokka T, Crama Y, Spieksma F (2014) Multi-dimensional vector assignment problems. Discrete Optim 14:111–125

Dunning I, Huchette J, Lubin M (2017) Jump: a modeling language for mathematical optimization. SIAM Rev 59(2):295–320

Ficker AMC, Spieksma FCR, Woeginger GJ (2018) Robust balanced optimization, KU Leuven, FEB Research report KBI_1802

Gabrel V, Murat C, Thiele A (2014) Recent advances in robust optimization: an overview. Eur J Oper Res 235(3):471–483

Galil Z, Schieber B (1988) On finding most uniform spanning trees. Discrete Appl Math 20(2):173–175

Gorissen BL, Yanıkoğlu İ, den Hertog D (2015) A practical guide to robust optimization. OMEGA 53:124–137

Kamura Y, Nakamori M (2014) Modified balanced assignment problem in vector case: System construction problem, In: 2014 international conference on computational science and computational intelligence (CSCI), vol 2. IEEE, pp 52–56

Katoh N, Iwano K (1994) Efficient algorithms for minimum range cut problems. Networks 24(7):395–407

Kinable J, Smeulders B, Delcour E, Spieksma F (2017) Exact algorithms for the equitable traveling salesman problem. Eur J Oper Res 261(2):475–485

Koster AM, Kutschka M, Raack C (2013) Robust network design: formulations, valid inequalities, and computations. Networks 61(2):128–149

Kouvelis P, Yu G (1997) Robust discrete optimization and its applications. Kluwer Academic Publishers, Norwell

Larusic J, Punnen A (2011) The balanced traveling salesman problem. Comput Oper Res 38(5):868–875

Lee C, Lee K, Park K, Park S (2012) Branch-and-price-and-cut approach to the robust network design problem without flow bifurcations. Oper Res 60(3):604–610

Martello S, Pulleyblank W, Toth P, De Werra D (1984) Balanced optimization problems. Oper Res Lett 3(5):275–278

Poss M (2014) Robust combinatorial optimization with variable cost uncertainty. Eur J Oper Res 237:836–845

Punnen A, Nair K (1999) Constrained balanced optimization problems. Comput Math Appl 37(9):157–163

Turner L (2012) Variants of shortest path problems. Algorithmic Oper Res 6(2):91–104

Wiesemann W, Kuhn D, Sim M (2014) Distributionally robust convex optimization. Oper Res 62(6):1358–1376