Partitioning Vectors into Quadruples: Worst-Case Analysis of a Matching-Based Algorithm

Annette M. C. Ficker

Faculty of Economics and Business, KU Leuven, Leuven, Belgium
Annette.Ficker@3ds.com

Thomas Erlebach[1]

Department of Informatics, University of Leicester, Leicester, United Kingdom
t.erlebach@leicester.ac.uk
https://orcid.org/0000-0002-4470-5868

Matúš Mihalák

Department of Data Science and Knowledge Engineering, Maastricht University, Maastricht, The Netherlands
matus.mihalak@maastrichtuniversity.nl
https://orcid.org/0000-0002-1898-607X

Frits C. R. Spieksma

Department of Mathematics and Computer Science, Eindhoven University of Technology, Eindhoven, The Netherlands
f.c.r.spieksma@tue.nl
https://orcid.org/0000-0002-2547-3782

Abstract

Consider a problem where $4k$ given vectors need to be partitioned into k clusters of four vectors each. A cluster of four vectors 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 problem is to partition the given $4k$ vectors into k quads with minimum total cost. We analyze a straightforward matching-based algorithm and prove that this algorithm is a $\frac{3}{2}$-approximation algorithm for this problem. We further analyze the performance of this algorithm on a hierarchy of special cases of the problem and prove that, in one particular case, the algorithm is a $\frac{5}{4}$-approximation algorithm. Our analysis is tight in all cases except one.

2012 ACM Subject Classification Mathematics of computing → Approximation algorithms

Keywords and phrases approximation algorithm, matching, clustering problem

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2018.45

Related Version A full version of the paper is available at [6], `https://arxiv.org/abs/1807.01962`.

[1] Supported by a study leave granted by University of Leicester.

29th International Symposium on Algorithms and Computation (ISAAC 2018).
Editors: Wen-Lian Hsu, Der-Tsai Lee, and Chung-Shou Liao; Article No. 45; pp. 45:1–45:12
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

Partitioning Vectors into Quadruples (PQ) is the problem of partitioning $4k$ given nonnegative vectors $v_1, \ldots, v_{4k}$, each consisting of n components, into k 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 the problem is to find a partition of the $4k$ vectors into k quads such that the total cost of all quads is minimum.

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 $|V| = 4k$). 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 in the complete graph G, yielding $2k$ vector pairs. Let $p_1, \ldots, p_{2k}$ be the $2k$ matched vector pairs corresponding to the computed matching M. 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 ($i = 1, \ldots, 2k$; $|V'| = 2k$). 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' in the complete graph G'. Each of the k edges of M' matches two vector pairs, which naturally induces a quad. The k quads induced by the edges of M' constitute a solution to the problem. Clearly, A is a polynomial-time algorithm. A rigorous description can be found in Section 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, and we are interested in analyzing how far off algorithm A's output can be from an optimum solution.

In this paper we show that A is a $\frac{3}{2}$-approximation algorithm for problem PQ, and that this bound is tight. We also show that algorithm A has better approximation guarantees for various special cases of problem PQ. In particular, consider an instance of PQ where each vector has exactly two ones, while all other components are zero. In that case, each vector can be seen as an edge in a graph where there is a node for each component. For the case where this graph is a simple, connected graph, we prove that A is a $\frac{5}{4}$-approximation algorithm. We give a precise overview of our results in Section 2.3.

The paper is organized as follows. The remainder of this section introduces some terminology and discusses related work that motivates our research. Section 2 gives preliminaries and states our results. In Section 3, we give the proof of the upper bound on the worst-case ratio of algorithm A for the special case of problem PQ mentioned above; we also outline the proofs for all other cases. Section 4 contains the lower bound result for the special case. We conclude in Section 5. Detailed proofs of all our bounds on the worst-case ratio of algorithm A for problem PQ and other generalizations of the special case can be found in [6].

1.1 Terminology and related literature

Worst-case analysis is a well-established tool to analyze the quality of solutions found by heuristics. We refer to books by Vazirani [13] and Williamson and Shmoys [14] for a thorough introduction to the field. We use the following, standard terminology that applies to minimization problems. In the next definition, $A(I)$ stands for the value of the solution to instance I found by algorithm A, while $OPT(I)$ stands for the value of an optimum solution to instance I.

▶ **Definition 1.** Algorithm A is an α-approximation algorithm for a minimization problem P if for every instance I of problem P: (i) algorithm A runs in polynomial time, and (ii) $A(I) \leq \alpha \cdot OPT(I)$. We refer to α as an upper bound on the worst-case ratio of algorithm A.

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 [10] consider a problem where a given set of vectors in n-dimensional space needs to be partitioned into 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 the 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 (cf. Section 2.1).

Another problem, distinct from, yet related to, our problem, comes from computational biology, and is described in Figuero et al. [7]. 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 [8] 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 $(\frac{13}{10} + \epsilon)$-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. [11] for a description). Central in this application is the production of so-called *waferstacks*, which can be seen as a set of superimposed wafers. In our context, a wafer can be represented by a vector. A wafer consists of many dies, 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 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. [4] 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. [3]; for a particular objective function, they describe a $\frac{4}{3}$-approximation algorithm.

A restricted, 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 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. [9]. Thomassen [12] studies the existence of edge-decompositions into paths of length 4, and Barat and Gerbner [1] even study edge-decompositions where each cluster is isomorphic to a tree consisting of 4 edges.

2 Preliminaries

2.1 About problem PQ: special cases and complexity

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

▶ **Lemma 2.** *Each instance of problem PQ with arbitrary (rational) nonnegative vectors can be reduced to an instance of problem PQ with $\{0,1\}$-vectors.*

The argument in the proof of Lemma 2 (see [6]) implies that any worst-case ratio of algorithm A shown to hold for instances consisting of $\{0,1\}$-vectors holds in fact for arbitrary rational nonnegative vectors. Clearly, this does not mean that algorithm A is restricted to work on instances consisting of binary vectors; it works directly on 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 the general 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($\#1 = 2, \text{distinct}$). 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($\#1 = 2, \text{distinct, connected}$). We distinguish a further special case by demanding that the graph F is also connected.

Clearly, the special cases are ordered, in the sense that each next one is a special case of its predecessor.

Although our interest is on the worst-case behavior of algorithm A, it is relevant to establish the computational complexity of problem PQ. It turns out (see [6] for the proof) that even its special case PQ($\#1 = 2, \text{distinct, connected}$) is NP-hard. This fact shows that no polynomial-time algorithm for problem PQ can be exact, unless P=NP.

▶ **Theorem 3.** *PQ($\#1 = 2$, distinct, connected) is NP-hard.*

2.2 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.:

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

Here, $v_{i,\ell}$ denotes the ℓ-th component of vector v_i ($\ell = 1, \ldots, n$). We use $|v_i|$ to denote the number of ones in vector v_i ($1 \leq i \leq 4k$), i.e., $|v_i| = \sum_{\ell=1}^{n} v_{i,\ell}$. 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_i and v_j, i.e.:

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

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

The following observation concerning two $\{0,1\}$-vectors u and v is immediate.

▶ **Observation 4.** $|u| + |v| = \text{sav}(u, v) + |u \vee v|$.

Let us revisit the description of Algorithm A. In the first phase, it computes a minimum-cost perfect matching M in the complete graph G on the given $4k$ vectors, where the weight of the edge between vectors v_i and v_j is set to $|v_i \vee v_j|$. Let $p_1, \ldots, p_{2k}$ be the $2k$ matched vector pairs corresponding to the computed matching M, and let $\text{cost}(M)$ denote the cost of the matching M. For $1 \leq i \leq 2k$, 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' in the complete graph G' on the $2k$ 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' are output as a solution. Let $\text{cost}(M')$ be the cost of matching M'.

▶ **Observation 5.** $A(I) = \text{cost}(M')$ *and* $\text{cost}(M') \leq \text{cost}(M)$.

▶ **Lemma 6.** *In the first phase of algorithm A, we can equivalently set the weight of the edge between v_i and v_j to be $-\text{sav}(v_i, v_j)$. Similarly, in the second phase of algorithm A, we can set the weight of the edge between p_i and p_j to be $-\text{sav}(v_i', v_j')$.*

Let $\text{weight}(M')$ denote the total savings of the perfect matching M', i.e., $\text{weight}(M') = \sum_{(v_i', v_j') \in M'} \text{sav}(v_i', v_j')$. Then, we have:

$$\text{cost}(M') = \text{cost}(M) - \sum_{(v_i', v_j') \in M'} \text{sav}(v_i', v_j') = \text{cost}(M) - \text{weight}(M'). \qquad (1)$$

Observation 5 and Equation (1) imply:

▶ **Corollary 7.** $A(I) = \text{cost}(M) - \text{weight}(M')$.

In view of this corollary, it follows that if we can show that $\text{cost}(M) \leq B$ and $\text{weight}(M') \geq S$ for some bounds B and S, we can conclude that $A(I) \leq B - S$.

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.

Table 1 Overview of bounds on the worst-case ratio of algorithm A. Proofs of the bounds marked with (*) are omitted due to space restrictions and can be found in [6].

Problem name	Lower Bound	Upper Bound
PQ	$\frac{3}{2}$	$\frac{3}{2}$ (*)
PQ($\#1 \in \{1,2\}$)	$\frac{3}{2}$ (*)	$\frac{3}{2}$
PQ($\#1 = 2$)	$\frac{4}{3}$ (*)	$\frac{4}{3}$ (*)
PQ($\#1 = 2$, distinct)	$\frac{5}{4}$	$\frac{13}{10}$ (*)
PQ($\#1 = 2$, distinct, connected)	$\frac{5}{4}$ (Observation 16)	$\frac{5}{4}$ (Lemma 14)

▶ **Lemma 8.** *There is a minimum-cost perfect matching in G, as well as in G', that contains a maximum number of identical pairs.*

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.

2.3 Our results

In this paper, we show the following bounds on the worst-case ratio of algorithm A (see Table 1 for a summary).

▶ **Theorem 9.** *Algorithm A is a $\frac{3}{2}$-approximation algorithm for problem PQ, and this bound is tight.*

▶ **Theorem 10.** *Algorithm A is a $\frac{3}{2}$-approximation algorithm for problem PQ($\#1 \in \{1,2\}$), and this bound is tight.*

▶ **Theorem 11.** *Algorithm A is a $\frac{4}{3}$-approximation algorithm for problem PQ($\#1 = 2$), and this bound is tight.*

▶ **Theorem 12.** *Algorithm A is a $\frac{13}{10}$-approximation algorithm for problem PQ($\#1 = 2$, distinct), and its worst-case ratio is at least $\frac{5}{4}$.*

▶ **Theorem 13.** *Algorithm A is a $\frac{5}{4}$-approximation algorithm for problem PQ($\#1 = 2$, distinct, connected), and this bound is tight.*

The proof of Theorem 13 is given in the next sections: the proof implying the upper bound (Lemma 14) is in Section 3.1, and the instance leading to the lower bound result (Observation 16) is in Section 4.1. In Section 3.2 we provide a high-level description of the proofs leading to the other upper bound results. Full proofs of all upper and lower bounds can be found in [6].

As an aside, we also give instances that show that the worst-case ratio of a natural greedy algorithm is worse than the worst-case ratio of algorithm A, both for problem PQ($\#1 = 2$, distinct, connected) (Section 4.2) and for problem PQ (see [6]).

3 Upper bound proofs

In this section, we prove that $\frac{5}{4}$ is an upper bound for the worst-case ratio of algorithm A for Problem PQ($\#1 = 2$, distinct, connected). The proofs for the upper bound $\frac{3}{2}$ for the worst-case ratio of Problem PQ, the upper bound $\frac{4}{3}$ for the worst-case ratio of Problem PQ($\#1 = 2$), and the upper bound $\frac{13}{10}$ for the worst-case ratio of Problem PQ($\#1 = 2$, distinct) can be found in [6]. An outline of our approach to derive these results is given in Section 3.2.

3.1 Approximation analysis for PQ($\#1 = 2$, distinct, connected)

▶ **Lemma 14.** *Algorithm A is a $\frac{5}{4}$-approximation algorithm for PQ($\#1 = 2$, distinct, connected).*

Proof. Recall that an instance of PQ($\#1 = 2$, distinct, connected) can be viewed as a simple, connected graph F with $4k$ edges, and that the cost of a quad is the number of vertices spanned by the edges in the quad. 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, $OPT \geq 4k$. Furthermore, if we can show that there are z quads in the optimal solution that have cost at least 5, we get that $OPT \geq 4(k - z) + 5z = 4k + z$.

▶ **Observation 15.** $\text{cost}(M) = 6k$.

Proof. The line graph of a connected graph with an even number of edges admits a perfect matching (Jünger et al. [9], Dong et al. [5]). Thus, the minimum-cost perfect matching M pairs adjacent edges of the graph. Hence, every pair in M has cost 3, and thus the cost of M is $2k \cdot 3 = 6k$. ◀

Let $p_1, \ldots, p_{2k}$ be the pairs corresponding to M. Consider the auxiliary graph H with vertex set $V' = \{p_1, \ldots, p_{2k}\}$ 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' 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, $1 \leq \mu \leq k$. 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

$$A(I) \leq 6k - \mu.$$

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

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

$$\frac{A(I)}{OPT(I)} \leq \frac{6k - \mu}{5k - \mu} \leq \frac{5}{4},$$

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

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

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

where $\text{odd}(H - X)$ 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 their incident edges, are removed from H). Hence, there exists a set $X \subseteq V'$ such that $\text{odd}(H - X) - |X| = 2k - 2\mu$. Let $d = \text{odd}(H - X)$, and let $O_1, O_2, \ldots, O_d$ denote the d odd components of $H - X$. We have

$$2k - 2\mu = d - |X|.$$

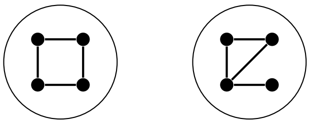

Figure 1 Quads with $\text{cost}(Q) = 4$.

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 $1 \leq i \leq d$ 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 $E_F(O_i)$, $1 \leq i \leq d$, and hence at least $2d$ 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:

$$2d \leq |SE4| + |SE5|. \tag{2}$$

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 by Q, as depicted in Figure 1. These four edges can be in the sets $E_F(O_i)$ for some $1 \leq i \leq d$, the set $E_F(X)$, and the sets $E_F(C)$ for even components C of $H - X$. 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 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 2. 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,

$$|E_F(X)| \geq |SE4|. \tag{3}$$

Further, since $|E_F(X)| = 2|X|$, it follows from (3) and (2) that $|SE5| \geq 2d - 2|X|$. Thus, the number of quads of cost at least 5 is at least $\frac{2d-2|X|}{4} = \frac{1}{2}(d - |X|) = k - \mu$. ◀

3.2 Outline of approximation analysis for other variants of PQ

In this section we give a high-level description of the crucial arguments we need to prove the three upper bound results for problems PQ, PQ($\#1 = 2$), and PQ($\#1 = 2$, distinct). As mentioned before, the full proofs are omitted due to space restrictions and can be found in [6].

To analyze algorithm A for PQ, we proceed along the following lines. By Corollary 7, we have $A(I) = \text{cost}(M) - \text{weight}(M')$. We fix an arbitrary optimal solution and define from

Table 2 Overview of different types of quads with cost 4, containing at least 1 edge from $E_F(O_i)$. The entry "1,1" for quad type 3 means that there is one edge from $E_F(O_i)$ and one edge from $E_F(O_{i'})$ for $i \neq i'$.

Type of quad	Number of edges in $E_F(O_i)$	in $E_F(X)$	in $E_F(C)$	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

it a perfect matching $\hat{M}$ in G and an amount of savings, written in the form $S_1 + \frac{1}{2}S_2$ for reasons explained below, that algorithm A can definitely achieve in the second phase. As $\text{cost}(M) \leq \text{cost}(\hat{M})$ and $\text{weight}(M') \geq S_1 + \frac{1}{2}S_2$, we have $A(I) \leq \text{cost}(\hat{M}) - (S_1 + \frac{1}{2}S_2)$.

The existence of the savings $S_1 + \frac{1}{2}S_2$ is shown by constructing a subgraph H of G' that is bipartite, has maximum degree 2, and in which each edge connects two vertices of the same degree. H consists of even-length cycles and isolated edges. Let S_2 be the total savings of the edges on cycles and S_1 the total savings of isolated edges in H. It follows that H contains a matching with total savings at least $S_1 + \frac{1}{2}S_2$, and thus G' contains a perfect matching with at least those savings.

The matching $\hat{M}$ and the graph H are determined by considering each quad Q of the optimal solution separately. For each quad $Q = \{v_1, v_2, v_3, v_4\}$ we define two vector pairs of $\hat{M}$ (by partitioning Q into two vector pairs in one of the three possible ways) and add to H either one edge (that becomes an isolated edge), or two edges (that will eventually be part of a cycle). For example, if the algorithm has matched $p = \{v_1, v_2\}$, $p_1 = \{v_3, v_3'\}$ and $p_2 = \{v_4, v_4'\}$ in M, where v_3' and v_4' are vectors not in Q, the edges added to H are (p, p_1) and (p, p_2). As another example, if the algorithm has matched $p_i = \{v_i, v_i'\}$ for $1 \leq i \leq 4$, we can show that we can add two disjoint edges of the form (p_i, p_j) for $i \neq j$ to H in such a way that H remains bipartite, and that there are two different ways of selecting these two edges.

In this way, each quad Q contributes an amount ϕ_Q to $\text{cost}(\hat{M}) - (S_1 + \frac{1}{2}S_2)$ that consists of the weight of the two edges it adds to $\hat{M}$ minus the savings of the edge it adds to H (if it adds only one isolated edge), or minus the savings of the two edges that it adds to H divided by two (otherwise). By selecting the edges added to $\hat{M}$ and H carefully among the valid possibilities, we can show that H has the desired properties and $\phi_Q \leq \frac{3}{2}\text{cost}(Q)$ holds for each quad Q of the optimal solution. Since $\text{cost}(\hat{M}) - (S_1 + \frac{1}{2}S_2) = \sum_Q \phi_Q$, this implies $A(I) \leq \frac{3}{2}OPT(I)$, showing that A is a $\frac{3}{2}$-approximation algorithm for problem PQ.

Now consider problem PQ($\#1 = 2$). Recall that the $4k$ input vectors can be viewed as edges in a multi-graph. Denote that multi-graph by F. To analyze algorithm A for PQ($\#1 = 2$), we follow the same approach as for PQ, but obtain the better bound $\phi_Q \leq \frac{4}{3}\text{cost}(Q)$ for each optimal quad Q by making a case distinction regarding the value of $\text{cost}(Q)$ and considering for each value of $\text{cost}(Q)$ all possible subgraphs of F that the edges of Q can induce. For example, if $\text{cost}(Q) = 3$, one of the cases is that the subgraph induced by Q is a 3-cycle with one duplicate edge. Assume that the four edges are $e_1 = (1, 2)$, $e_2 = (1, 2)$, $e_3 = (1, 3)$ and $e_4 = (2, 3)$. By Lemma 8, we can assume that algorithm A has matched e_1 with 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 $\hat{M}$, with total cost $2 + 3 = 5$. Consider the case that p_2 was not matched by A in the

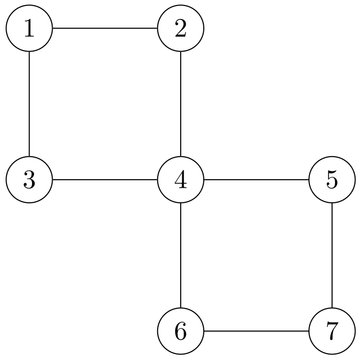

Figure 2 An instance of $\mathrm{PQ}(\#1 = 2, \text{distinct, connected})$.

first phase. (This is the more difficult case.) Assume that A has matched $p_3 = \{e_3, x\}$ and $p_4 = \{e_4, y\}$, where x and y are edges not in Q. We add (p_1, p_3) and (p_1, p_4) to H. Each of these edges has savings at least 1, and thus they contribute 2 to S_2, or 1 to $\frac{1}{2}S_2$. We have $\phi_Q \leq 5 - 1 = 4 = \frac{4}{3}\mathrm{cost}(Q)$. As $\phi_Q \leq \frac{4}{3}\mathrm{cost}(Q)$ can be shown to hold also for all other cases of quads Q in the optimal solution, algorithm A is a $\frac{4}{3}$-approximation algorithm for $\mathrm{PQ}(\#1 = 2)$.

For problem $\mathrm{PQ}(\#1 = 2, \text{distinct})$, the $4k$ input vectors can be viewed as the edges of a simple graph. We follow the same approach as in the previous paragraph, but since a simple graph with four edges spans at least 4 nodes, we only need to consider cases where $\mathrm{cost}(Q) \geq 4$. This allows us to show that $\phi_Q \leq \frac{13}{10}\mathrm{cost}(Q)$ in all cases, implying that algorithm A is a $\frac{13}{10}$-approximation algorithm for this problem.

4 Bad instances

In Section 4.1 we provide an instance that, together with the result in the previous section, yields the tight bound claimed for problem $\mathrm{PQ}(\#1 = 2, \text{distinct, connected})$ in Theorem 13. We illustrate in Section 4.2 that a natural greedy algorithm (that can be seen as an alternative for algorithm A) has a worst-case ratio worse than the worst-case ratio of algorithm A. The instances that provide lower bound results for problem PQ and the other special cases, as announced in Table 1, can be found in [6].

4.1 An instance of $\mathrm{PQ}(\#1 = 2, \text{distinct, connected})$

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 $\mathrm{PQ}(\#1 = 2\}, \text{distinct, connected})$. The instance can be represented by the graph shown in Figure 2.

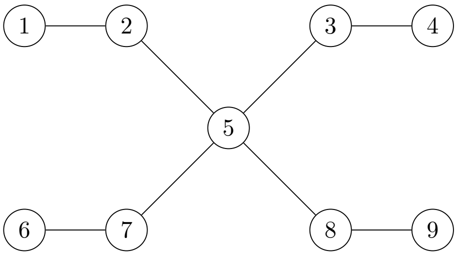

Figure 3 An instance of PQ($\#1 = 2$, distinct, connected).

The optimal solution for this instance has cost 8, 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 7 we have $A(I) \geq 10$. Hence, the worst-case ratio of A is at least $10/8 = 5/4$.

▶ **Observation 16.** *For the instance depicted in Figure 2, cost*$(A) = \frac{5}{4} OPT$.

Theorem 13 now follows from Lemma 14 and Observation 16.

4.2 Bad instances for a natural greedy algorithm

In this section, we show that the worst-case ratio of a natural greedy algorithm is worse than the worst-case ratio of algorithm A.

An informal description of the 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 an instance of problem PQ($\#1 = 2$, distinct, connected) showing that the worst-case performance of this greedy algorithm is worse than the worst-case performance of algorithm A. In [6] we present an instance of problem PQ with the same property.

An instance of PQ($\#1 = 2$, distinct, connected)

Consider the instance I of PQ($\#1 = 2$, distinct, connected) consisting of 8 vectors represented in a graph shown in Figure 3 (recall that a vector in PQ($\#1 = 2$, distinct,connected) corresponds to an edge in a simple 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: $\{(1,2), (3,4), (6,7), (8,9)\}$.

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.

5 Conclusion

We have studied the worst-case behavior of a natural algorithm for partitioning a given set of vectors into quadruples and shown the precise worst-case behavior of this algorithm for all cases except $\mathrm{PQ}(\#1 = 2, \mathrm{distinct})$, where a small gap remains. It is a natural question to study an extension where we form clusters consisting of 2^s vectors for some given integer $s \geq 2$. 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 be the minimum-cost matching of the first round. Then $A(I) \leq \mathrm{cost}(M)$ and $OPT(I) \geq \mathrm{cost}(M)/2^{s-1}$ as the cost of the optimum (viewed as being constructed in s rounds) is at least $\mathrm{cost}(M)$ 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 $\frac{3}{2}$ times the optimal cost after two rounds, we get a ratio of $\frac{3}{2} \times 2^{s-2} = 3 \times 2^{s-3}$. We leave the question of finding the worst-case ratio for arbitrary s as an open problem.

References

1. J. Barát and D. Gerbner. Edge-Decomposition of Graphs into Copies of a Tree with Four Edges. *The Electronic Journal of Combinatorics*, 21(1):1–55, 2014.
2. C. Berge. Sur le couplage maximum d'un graphe. *Comptes Rendus de l'Académie des Sciences*, 247:258–259, 1958.
3. T. Dokka, M. Bougeret, V. Boudet, R. Giroudeau, and F.C.R. Spieksma. Approximation algorithms for the wafer to wafer integration problem. In *Proceedings of the 10th International Workshop on Approximation and Online Algorithms (WAOA 2012)*, volume 7846 of *LNCS*, pages 286–297. Springer, 2013.
4. T. Dokka, Y. Crama, and F.C.R. Spieksma. Multi-dimensional vector assignment problems. *Discrete Optimization*, 14:111–125, 2014.
5. F. Dong, W. Yan, and F. Zhang. On the number of perfect matchings of line graphs. *Discrete Applied Mathematics*, 161(6):794–801, 2013.
6. Annette M. C. Ficker, Thomas Erlebach, Matús Mihalák, and Frits C. R. Spieksma. Partitioning Vectors into Quadruples: Worst-Case Analysis of a Matching-Based Algorithm. *CoRR*, abs/1807.01962, 2018. `arXiv:1807.01962`.
7. A. Figueroa, A. Goldstein, T. Jiang, M. Kurowski, A. Lingas, and M. Persson. Approximate clustering of fingerprint vectors with missing values. In *Proceedings of the 2005 Australasian Symposium on Theory of Computing (CATS 2005)*, volume 41 of *CRPIT*, pages 57–60. Australian Computer Society, 2005.
8. D.S. Hochbaum and A. Levin. Covering the edges of bipartite graphs using $\mathrm{K}_{2,2}$ graphs. *Theoretical Computer Science*, 411(1):1–9, 2010.
9. M. Jünger, G. Reinelt, and W.R. Pulleyblank. On partitioning the edges of graphs into connected subgraphs. *Journal of Graph Theory*, 9(4):539–549, 1985.
10. S. Onn and L.J. Schulman. The vector partition problem for convex objective functions. *Mathematics of Operations Research*, 26(3):583–590, 2001.
11. S. Reda, G. Smith, and L. Smith. Maximizing the functional yield of wafer-to-wafer 3-D integration. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 17(9):1357–1362, 2009.
12. C. Thomassen. Edge-decompositions of highly connected graphs into paths. In *Abhandlungen aus dem Mathematischen Seminar der Universität Hamburg*, volume 78, pages 17–26. Springer, 2008.
13. V.V. Vazirani. *Approximation Algorithms*. Springer-Verlag, Inc., New York, USA, 2001.
14. D.P. Williamson and D.B. Shmoys. *The Design of Approximation Algorithms*. Cambridge University Press, New York, USA, 1st edition, 2011.