ELSEVIER

Available online at www.sciencedirect.com
SCIENCE DIRECT®

Discrete Applied Mathematics 150 (2005) 121–139

DISCRETE APPLIED MATHEMATICS

www.elsevier.com/locate/dam

On uniform *k*-partition problems☆

Paolo Dell'Olmo[a], Pierre Hansen[b], Stefano Pallottino[c], Giovanni Storchi[d]

[a]*Dipartimento di Statistica, Probabilità e Statistiche Applicate, Università di Roma "La Sapienza", P.le Aldo Moro 5, 00185 Roma, Italy*
[b]*GERAD and École des Hautes Etudes Commerciales, 3000, Chemin de la Côte-Sainte-Catherine Montréal, Qué., Canada H3T 2A7*
[c]*Dipartimento di Informatica, Università di Pisa, Via F. Buonarroti 2, 56127 Pisa, Italy*
[d]*Dipartimento di Statistica, Probabilità e Statistiche Applicate, Università di Roma "La Sapienza", P.le Aldo Moro 5, 00185 Roma, Italy*

Received 3 January 2002; received in revised form 28 October 2004; accepted 15 February 2005
Available online 17 May 2005

This paper is dedicated to the memory of our friend and colleague Stefano Pallottino, disappeared prematurely on the April 11, 2004

Abstract

We study various uniform *k*-partition problems which consist in partitioning *m* sets, each of cardinality *k*, into *k* sets of cardinality *m* such that each of these sets contains exactly one element from every original set. The problems differ according to the particular measure of "set uniformity" to be optimized. Most problems are polynomial and corresponding solution algorithms are provided. A few of them are proved to be NP-hard. Examples of applications to scheduling and routing problems are also discussed.

Keywords: Partition; Matrix permutation; Algorithms; Complexity

E-mail addresses: paolo.dellolmo@uniroma1.it (P. Dell'Olmo), Pierre.Hansen@gerad.ca (P. Hansen), giovanni.storchi@uniroma1.it (G. Storchi).

☆ Research partially supported by grants: FCAR of Québec, NSERC of Canada, MIUR-SORSA, INDAM-GNAMPA, CNR-Agenzia 2000 contract n.CNRC00AF27-001 of Italy.

0166-218X/$ - see front matter © 2005 Elsevier B.V. All rights reserved.
doi:10.1016/j.dam.2005.02.013

1. Introduction

Let $R = \{R_1, \ldots, R_m\}$ be a collection of m sets. Any set $R_i = \{w_{i1}, \ldots, w_{ik}\}$ contains exactly k real values, i.e., $|R_i| = k$, $i = 1, \ldots, m$. The k-partition problem is to partition R into k sets $\{C_1, \ldots, C_k\}$ such that $|C_j \cap R_i| = 1$ for every pair i, j, i.e., every set C_j, $j = 1, \ldots, k$, contains exactly one element of any R_i, $i = 1, \ldots, m$. As a consequence $|C_j| = m$, $j = 1, \ldots, k$. We denote by $n = mk$ the global number of elements.

The problem can be more easily described in terms of matrix permutations. That is, let $W = [w_{ij}]$ be a real-valued matrix with m rows and k columns obtained by considering the sets $\{R_1, \ldots, R_m\}$ as its rows, and let $\Pi(W)$ be the set of all matrices obtained by permuting elements in the rows of W. Any permutation $\pi(W) \in \Pi(W)$ of values in each row generates a new matrix $W' = \pi(W)$ whose columns correspond to a k-partition $\{C_1, \ldots, C_k\}$.

In the following, we present k-partition problems as row-permutation problems of matrix W. More formally, given a matrix W and a permutation of elements $\pi_i(W)$ for each row i, we say that the columns of matrix $W' = \pi(W)$ are a *k-partition* of the rows of W, or, for brevity, that W' is a *k-partition* of W.

Various measures can be adopted to evaluate the k-partition. They consist in both an *inner measure* among the elements belonging to the same column and a *global measure* among the inner measures associated to the columns. We adopt four different measures, namely the *minimum*, the *maximum*, the *range*, and the *sum*. By combining these four measures at the inner and at the global levels, and by either maximizing or minimizing the global measure, 32 different k-partition problems are obtained.

The objective of this paper is to assess the computational complexity of each problem, providing solution algorithms for polynomial cases and NP-completeness proofs for intractable ones; however, the classification of one problem remains open. For convenience, we group the problems into classes according to their solution characteristics and computational complexity. Although the paper is mainly theoretical, we also discuss examples in which some of the problems have practical applications.

The paper is organized as follows. In Section 2, we introduce the notation, the problem definition and formalize the different measures (or objective functions). In Section 2, we show which cases are trivial, while Section 3 addresses all other polynomial cases. The NP-hard ones are presented in Section 4. In Section 5, we discuss an open problem. Finally, in Section 6, we propose several applications of the k-partition problems and suggest further research lines.

2. Notation and problem definition

Let $\pi_i(W)$ be a permutation among the elements of row i of matrix W, $i = 1, \ldots, m$; by $\pi(W) = [\pi_i(W)]$ we denote such permutations for all m rows of W. Let $W' = \pi(W)$ be the *permuted matrix*; we denote by w'_{ij} the ith element of column C_j of W' obtained by permutation $\pi_i(W)$. The four different *inner measures* of the k columns $\{C_1, \ldots, C_k\}$ of W' are defined and noted as follows:

$$u_j = u(C_j) = \max\{w'_{ij} : i = 1, \ldots, m\} \tag{1}$$

and

$$l_j = l(C_j) = \min\{w'_{ij} : i = 1, \ldots, m\} \tag{2}$$

are the *maximum* and the *minimum* of column C_j, $j = 1, \ldots, k$, respectively; the *range* of C_j, $j = 1, \ldots, k$, is

$$\delta_j = \delta(C_j) = u_j - l_j, \tag{3}$$

while σ_j denotes the *sum* of (elements of) C_j:

$$\sigma_j = \sigma(C_j) = \sum_{i=1}^{m} w'_{ij}. \tag{4}$$

We use the generic form f_j to indicate one of the four above-mentioned inner measures for C_j, i.e. the jth column of W'. The four *global measures* among the columns of matrix W' are defined and noted as follows:

$$U = U(W') = \max\{f_j : j = 1, \ldots, k\}, \tag{5}$$

$$L = L(W') = \min\{f_j : j = 1, \ldots, k\}, \tag{6}$$

$$\Delta = \Delta(W') = U - L, \tag{7}$$

$$\Sigma = \Sigma(W') = \sum_{j=1}^{k} f_j, \tag{8}$$

are the *maximum*, the *minimum*, the *range* and the *sum* of W', respectively.

If we indicate by F the generic global measure function among the four introduced above, we have 16 different k-partition measures. We want either to maximize or to minimize F so we obtain 32 different optimization problems.

To denote one of these problems we will use:

- "*max*" and "*min*" to stress the maximization and the minimization of the objective function;
- the symbols l, u, δ and σ for the inner measure;
- the symbols L, U, Δ and Σ for the global measure.

For example, maximizing the range (7) among the columns, where the inner measure is the minimum (2), will be denoted by $max(l, \Delta)$, that is to find an optimal matrix W^* such that $z = z(W^*) = \max\{\Delta(W') : W' \in \Pi(W)\}$, where $\Delta(W')$ is defined in (7).

To better clarify the difficulty of each problem, we will give its time complexity together with the number of *exchange* operations needed to permute W in order to *provide* W', that is the number of swappings of pairs of elements.

In Tables 1 and 2 we give a global view of the computational complexities, separately for *max* and *min*. The question mark in Table 2 indicates that complexity of problem $min(\delta, \Delta)$ remains open.

Some of the above problems admit a trivial solution. They are nevertheless considered for completeness.

Table 1
Complexity of the maximization problems

max	L	U	Δ	Σ
l	n	n	n	$n \log k$
u	n	n	n	n
δ	$k^{2.5}$	n	$n \log n$	n
σ	NP-hard	n	n	n

Table 2
Complexity of the minimization problems

min	L	U	Δ	Σ
l	n	n	n	n
u	n	n	n	$n \log k$
δ	$n \log n$	$n \log k$	**?**	$n \log k$
σ	n	NP-hard	NP-hard	n

2.1. Easy problems

This class contains all the problems for which the objective function takes a constant value for any $W' \in \Pi(W)$ and all partitions (except in one case) are optimal; so, the input matrix W can be seen as the permuted matrix W' without any swapping of elements. Consider the problems:

- $P1$: $min(u, U)$,
- $P2$: $max(u, U)$.

For both of them the optimal value is $z = w_{max} = \max\{w_{ij} : i = 1, \ldots, m, j = 1, \ldots, k\}$, i.e., a constant value which can be easily found by inspection in $O(n)$ time; obviously W is optimal.

Similarly, for the problems:

- $P3$: $min(l, L)$,
- $P4$: $max(l, L)$;

the optimal value is $z = w_{min} = \min\{w_{ij} : i = 1, \ldots, m, j = 1, \ldots, k\}$.

On the other hand, the optimal value z is given by $z = \sum_{i=1}^{m} \sum_{j=1}^{k} w_{ij}$ for the following problems:

- $P5$: $min(\sigma, \Sigma)$,
- $P6$: $max(\sigma, \Sigma)$;

and this sum is computed in $O(n)$ time.

Consider now the problem:

- $P7$: $max(\delta, U)$;

since we want to maximize the largest range of the columns, this value is obtained by having w_{max} and w_{min} in the same column. This is an easy task once the position of these two elements in W is known, which takes $O(n)$ time. To obtain the permuted matrix it suffices to move one of them to the column containing the other.

Only one special case has to be considered, i.e., when these two elements belong to the same row; in that case it is easy to prove that the largest range is defined either by w_{max} and the smallest element not belonging to the same row, or by the w_{min} and the largest element not belonging to the same row (more precisely, by the greatest among these two ranges). Again this takes $O(n)$ time and the permuted matrix W' is obtained from W through at most one exchange operation.

2.2. Grouping maximum and/or minimum row elements

The following three problems have as permuted matrix W' any matrix derived from W by grouping the maximum elements of each row in the same column:

- $P8$: $max(\sigma, U)$,
- $P9$: $max(l, U)$,
- $P10$: $max(l, \Delta)$.

In problem $P8$ we want a column whose sum of elements is as large as possible. Such a column is, indeed, that one containing the maximum element of each row. Finding these elements takes $O(n)$ time, and, with at most m exchanges, W is transformed into W'.

Analogously, for problem $P9$, we want to maximize the largest amongst the column minima, see (2) and (5):

$$\max\{\max\{l(C_j) : j = 1, \ldots, k\} : W' \in \Pi(W)\}.$$

Thus, it is sufficient to build a permutation with one column having its minimum as large as possible, that is a column with the maximum of row i as the ith element, for all i, regardless of the other columns. The same permuted matrix solves problem $P10$, where the range (7) between the minima of the columns has to be maximized:

$$\max\{l_j - l_h : j, h = 1, \ldots, k\} = \max\{l_j : j = 1, \ldots, k\} - \min\{l_h : h = 1, \ldots, k\}.$$

Since $\min\{l_h : h = 1, \ldots, k\} = w_{min}$ is a constant, problem $P10$ reduces to $P9$.

The following three problems are solvable in the same way as the previous three, with minimum row values instead of the maximum ones:

- $P11$: $min(\sigma, L)$,
- $P12$: $min(u, L)$,
- $P13$: $max(u, \Delta)$.

Let us analyze now the following problem:

- $P14$: $max(\sigma, \Delta)$;

its objective function value is:

$$\max\{\sigma_j - \sigma_h : j, h=1, \ldots, k\} = \max\{\sigma_j : j=1, \ldots, k\} - \min\{\sigma_h : h=1, \ldots, k\}.$$

Thus, the maximum value of the objective function is obtained by grouping the maximum row elements in one column and the minimum ones in another. So, problem $P14$ can be solved by selecting, also in $O(n)$ time, the extreme values of each row and by grouping these values in two specific columns with at most $2m$ exchanges.

3. Other polynomial problems

The following polynomial problem are grouped in four different classes.

3.1. Grouping elements belonging to a given interval

Let us consider the following problems:

- $P15$: $min(\delta, L)$,
- $P16$: $max(\delta, \Delta)$.

In problem $P15$ we want to minimize the minimum among the column ranges (3), this is a balanced optimization problems considered by Martello et al. [9].

Let us call a real interval $[a, b]$ *usable* for matrix W if there exists at least one element in each row of W belonging to that interval.

Clearly, if $[a, b]$ is usable, it is possible to select in each row one element belonging to that interval and to group the selected elements in the same column, say $\bar{j}$. Consider the permuted matrix W' so obtained; the range $\delta_{\bar{j}}$ is not greater than the interval width $\bar{\delta}=b-a$; so, $\bar{\delta}$ is an upper bound for $z(W') = \min\{\delta_j : j = 1, \ldots, k\}$ and also for the optimal value z among all permuted matrices.

A given usable interval of width $\bar{\delta}$ is said to be *minimal* for W if no usable interval with smaller width exists. Note that the width $\bar{\delta}$ of a minimal interval is the optimal value of the objective function for problem $P15$. In fact, the permuted matrix W' obtained by grouping the selected elements in the same column $\bar{j}$, has exactly $\bar{\delta}$ as the minimum range. To prove the above statement it is sufficient to observe that neither column $\bar{j}$ nor the other ones can have a range smaller than $\bar{\delta}$, since the interval is minimal. For the same reason, we can conclude that W' is optimal. Note that, once the minimal interval is known, only m exchanges are necessary to provide W'.

Let us suppose now that it is possible to construct a finite sequence of usable intervals having the property that at least one of them is minimal. Such a sequence will be called a *feasible sequence*.

In the following we prove that it is possible to build a feasible sequence with no more than n intervals in $O(n \log n)$ time. So, the complexity of problem $P15$ is $O(n \log n)$.

Let T be the set of values of the components w_{ij} of W, $|T| = t \leqslant n$; since the aim is to build usable intervals of minimum width, the following observations are easy to prove:

1. only intervals whose extreme values belong to T must be considered;
2. between two usable intervals $[a', b]$ and $[a'', b]$ such that $a' < a''$ only the latter must be considered; similarly, among intervals $[a, b']$ and $[a, b'']$ with $b' < b''$ only the former must be considered;
3. if an interval $[a, b]$ is not usable, by denoting with I the index set of rows without elements belonging to $[a, b]$, the minimum width usable interval $[a, b']$ having a as lower bound is such that

$$b' = \max\{\min\{w_{ij} : w_{ij} > b,\ j = 1, \ldots, k\} : i \in I\},$$

   under the condition that for each $i \in I$ there exists at least one $w_{ij} > b$; otherwise, no usable intervals with lower bound $\geqslant a$ exist;
4. if $[a, b]$ is usable, the minimum width usable interval $[a', b]$ having b as upper bound is such that

$$a' = \min\{\max\{w_{ij} : w_{ij} \leqslant b,\ j = 1, \ldots, k\} : i = 1, \ldots, m\};$$

5. given a usable interval $[a, b]$ such that $[a', b]$ is not usable for any $a < a' \leqslant b$, the next lower bound to be considered in a feasible sequence is

$$a'' = \min\{w_{ij} : w_{ij} > a,\ i = 1, \ldots, m,\ j = 1, \ldots, k\}.$$

By using the above observations it is possible to devise a procedure to build a feasible sequence. Starting from a suitable value for a, through observation 3 the upper bound b' is found and, through observation 4 the lower bound a' is detected such that $[a', b']$ is a usable interval of the sequence, and of minimal width with respect to b'. The next value a to be used iteratively to build the feasible sequence is obtained as described in observation 5. In the procedure, the minimum width current interval $[\bar{a}, \bar{b}]$ is maintained. At the end, i.e., when applying observation 3 no further usable intervals can be detected, $[\bar{a}, \bar{b}]$ is a minimal interval and $\bar{\delta} = \bar{b} - \bar{a}$ is the optimal value for $P15$.

The following lemma guarantees that the feasible sequence built by the above procedure is limited by n.

Lemma 1. *The maximum number of intervals built by the procedure is $t \leqslant n$.*

Proof. Each time a new interval is built, its lower bound is strictly greater than the lower bound of the previous interval. Since the number of different w_{ij} values is t, the result follows. □

Let $Q = \{[a_1, b_1], [a_2, b_2], \ldots, [a_q, b_q]\}$ with $q \leqslant t$, be the feasible sequence resulting from the above procedure, such that $a_h < a_{h+1}$, for each $h = 1, \ldots, q - 1$.

Let us suppose that we have found the first h intervals of Q. By applying observation 5 to $[a_h, b_h]$ we obtain the value a'' such that $[a'', b_h]$ is not usable, through observation 3 we

obtain $b_{h+1} = b'$ (if it exists) and, finally, through observation 4 we obtain $a_{h+1} = a'$, and so the next interval $[a_{h+1}, b_{h+1}]$.

The following result is easy to prove:

Theorem 1. *The interval* $[a_{h+1}, b_{h+1}]$ *obtained by the above procedure is such that the following properties hold*:

(i) $[a_{h+1}, b_{h+1}]$ *is usable*,
(ii) $a_h < a_{h+1}$ *and* $b_h < b_{h+1}$,
(iii) *no usable interval* $[a, b] \subset [a_{h+1}, b_{h+1}]$ *exists.*

Proof. The formulas used in observations 3 and 4 guarantee that, for each row $i = 1, \ldots, m$, at least one value w_{ij} exists such that $w_{ij} \in [a_{h+1}, b_{h+1}]$, and hence the first property holds. Since the interval $[a'', b_h]$ obtained through observation 5 is not usable, observation 3 ensures that $b_{h+1} = b' > b_h$, and observation 4 ensures that $a_{h+1} = a' \geqslant a'' > a_h$; thus, the second property also holds.

As far as the third property is concerned, if by contradiction we suppose that such an usable interval $[a, b]$ exists, we obtain that either $a > a_{h+1} = a'$ or $b < b_{h+1} = b'$, or both. In any case, we contradict the hypotheses that a' and b' are equal to the minimum and maximum possible values, according to observations 4 and 3, respectively. □

Let us now formalize the above scheme into an algorithm in order to evaluate its complexity.

The rows $\{R_1, \ldots, R_m\}$ of matrix W are properly sorted, one at a time through procedure *Sort*(i), in non decreasing order of their values. This preprocessing takes $O(k \log k)$ time for each row and, globally, $O(n \log k)$ time.

To each row i, for $i = 1, \ldots, m$, a *pointer* $j(i)$ is properly maintained to indicate the highest index of the sorted row R_i in which the corresponding element $w_{ij(i)}$ belongs to the current interval, i.e., it is not greater than the current upper bound b_h:

$$j(i) = \arg\max\{w_{ij} : j = 1, \ldots, k, w_{ij} \leqslant b_h\}.$$

A binary heap, of size m, contains m rows $\{R_1, \ldots, R_m\}$; the *key* associated to row R_i is $w_{ij(i)}$. A minimum key row is at the root of the heap. As shown in the following, the row pointers can only increase and, since the rows have been sorted, their keys cannot decrease. The role of the heap is to select the minimum value a' (observation 4), to remove all the minimum values until value a'' is found (observation 5), and to implicitly build the set I in order to apply observation 3.

It is easy to prove that the first value b_1 is $b_1 = \widehat{w} = \max\{w_{i1} : i = 1, \ldots, m\}$, i.e., it is the highest among the row minima. It is also easy to prove that the last value of the lower bound is $a_q = \bar{w} = \min\{w_{ik} : i = 1, \ldots, m\}$, i.e., it is the lowest among the row maxima.

In the algorithm *Interval*, we represent by *EmptyHeap*, *AddHeap*(i), *MinHeap*(i), and *UpdateHeap*(i), the basic operations for building an empty heap of size m, for adding element i (together with its current key $w_{ij(i)}$) to it, for retrieving the minimum key element i, and for updating the heap after the change of the key of element i, respectively.

```
Procedure Interval:
    begin {initialization}
        Sort rows R_1, ..., R_m; compute ŵ and w̄;
        ā := w_min; b̄ := b_1 := ŵ; δ̄ := b̄ − ā; h := 1; EmptyHeap;
        for i := 1 to m do
            begin {initializing the heap}
                while j(i) < k and w_{ij(i)+1} ⩽ b_1 do j(i) := j(i) + 1;
                AddHeap(i)
            end;
        repeat {main loop}
            MinHeap(i); a_h := w_{ij(i)};
            if b_h − a_h < δ̄ then begin ā := a_h; b̄ := b_h; δ̄ := b̄ − ā end;
            if a_h < w̄ then
                begin {starting for a new interval by setting b_h}
                    b_{h+1} := b_h; h := h + 1;
                    repeat {updating the pointer of every minimum key row i}
                        j(i) := j(i) + 1; UpdateHeap(i);
                        if w_{ij(i)} > b_h then b_h := w_{ij(i)}; MinHeap(i)
                    until w_{ij(i)} > a_{h−1};
                    for i := 1 to m do
                        begin {updating the row pointers according to the new b_h}
                            while j(i) < k and w_{ij(i)+1} ⩽ b_h do j(i) := j(i) + 1;
                            UpdateHeap(i)
                        end
                end
        until a_h = w̄;
        return {ā, b̄, δ̄}
    end.
```

In the initialization, the two values $\hat{w}$ and $\bar{w}$ are computed and a first interval $[\bar{a}, \bar{b}]$ is assigned to initialize $\bar{\delta}$. In the first loop, the key of each row is properly assigned according to the initial value b_1, and the first heap is built.

In the main loop, the minimum key gives the lower bound a_h, and the search of a new interval starts, after having possibly updated the current minimum interval $[\bar{a}, \bar{b}]$, only if a_h did not reach the maximum possible value $\bar{w}$.

In the inner **repeat …until** loop, for each row i, whose key is a minimum one (i.e. $w_{ij(i)} = a_{h-1}$), the new key is selected and the upper bound is updated. Once the value of b_h is established, the keys of the rows are properly updated in order to have the minimum key row at the root of the heap; this minimum key gives the new lower bound a_h.

At the end, the minimum interval $[\bar{a}, \bar{b}]$ and its width $\bar{\delta}$ are returned; by grouping elements belonging to that interval in the same column the optimal permuted matrix of problem $P15$ is obtained.

Let us now analyze the complexity of the algorithm *Interval*. The rows sorting takes $O(n \log k)$ time, while the initial heap is obtained in $O(n + m \log m)$ time, since m insertions into the heap of size m are performed and no more than n scannings are necessary to set the row keys.

The main loop is repeated $q \leqslant t \leqslant n$ times. To evaluate the complexity, it is easier to analyze the cost of the operations globally.

The first selection of the heap minimum, the updating of the current minimum interval and the initialization of the upper bound cost globally $O(q) = O(n)$ time. Still globally, the inner loop costs $O(n \log m)$ time since no more than n changes of keys are possible. The last **for …do** loop has the same time complexity $O(n \log m)$; in fact, globally, no more than n key changes are possible, and if the key is not changed, the heap updating is performed in constant time.

The above considerations prove the following result.

Theorem 2. *Algorithm* Interval *is correct and runs in* $O(n \log n)$ *time in worst case.*

Proof. The correctness of the algorithm has been already proved in the description of the algorithm's behavior. To establish its complexity, note that presorting the rows costs $O(n \log k)$ time, while building the feasible sequence of intervals and finding those with minimum width costs $O(n \log m)$ time. Since the highest among the two logarithms is bounded from above by $O(\log n)$, the result follows. □

As far as problem $P16$ is concerned, we recall that its objective function, to be maximized, is $\max\{\delta_j : j=1, \ldots, k\} - \min\{\delta_j : j=1, \ldots, k\}$. To maximize the first part is equivalent, as shown for problem $P7$, to grouping the maximum and the minimum matrix elements (with a small exception in a particular case) in the same column, while to minimize the second part is equivalent to problem $P15$. It is not difficult to combine the two procedures and to consider the possible exceptions. These latter are: maximum and minimum belonging to the same row; elements to be grouped in the "interval" column are the "max–min" elements for the other column. It is easy to verify that the number of exceptions is constant, and hence, $P16$ has the same time complexity as $P15$.

3.2. Spreading k elements on k columns

Consider the following three problems:

- $P17$: $max(u, L)$,
- $P18$: $max(u, \Sigma)$,
- $P19$: $min(u, \Delta)$;

to solve them it is sufficient to find the k biggest elements in the whole matrix W and permute them such that they belong to different columns of W'. Finding the kth biggest element among n can be solved in $O(n)$ time [4]; then, also in $O(n)$ time, the k biggest elements can be retrieved. Once the positions of these elements is known, the *spreading phase*, to ensure that no pair of them belong to the same column, requires k exchanges at most and can be done in $O(k)$ time.

In fact, since problem $P17$ requires that the smallest maximum value among columns be as big as possible, to assign one of the k biggest elements to each column guarantees that the objective function value is exactly the kth biggest value. The same holds for problem $P18$ in which maximizing the sum of the maximum column value is the objective function. In problem $P19$, we want to minimize the difference between the biggest and the smallest column maximum values; since the first is a constant, this is equivalent to maximizing the smallest maximum column value, which is indeed the kth biggest one.

The following problems are symmetric to the previous ones, but they are based on the k smallest values and on spreading them in different columns:

- $P20$: $min(l, U)$,
- $P21$: $min(l, \Sigma)$,
- $P22$: $min(l, \Delta)$.

As far as the following problem is concerned:

- $P23$: $max(\delta, \Sigma)$;

the objective function to maximize is the sum of the column ranges (3):

$$\sum_{j=1}^{k} \delta_j = \sum_{j=1}^{k} u_j - \sum_{j=1}^{k} l_j.$$

The maximum is obtained by spreading in different columns both the k biggest and the k smallest elements of matrix W; in fact, the first sum is maximized and the second one is minimized. When permuting elements, we have to take care when moving the already spread biggest elements during the spreading phase of the smallest ones. The permutation complexity is still $O(k)$ time.

3.3. Spreading k pairs of elements on k columns

In this subsection we analyze the following problem:

- $P24$: $max(\delta, L)$.

Let us assume we know a lower bound lb of the optimum value of the objective function; i.e., a value which is not greater than the column ranges of an optimal permuted matrix W^*. Then, in each column C_j of the permuted matrix W' there will exist at least two values, which we call u'_j and l'_j, such that

$$u'_j - l'_j \geqslant lb, \quad j = 1, \ldots, k. \tag{9}$$

Given a column h and a value u'_h belonging to it, in order to satisfy (9), one can select the minimum value element among those that can be paired with u'_h. Note that the $k-1$ w_{ij} which belong to the same row as u'_h cannot be grouped in column h at the same time as u'_h; moreover, in the worst case, the $k-1$ smallest w's can be used as elements l'_j in the columns of index $j \neq h$. For this reason we can state that in finding the "mate" element l'_h it is sufficient to analyze the $g = 2k-1$ smallest elements of W, which form the *ground set GD*. This set is valid for every column $j = 1, \ldots, k$. By symmetry, we consider also

the *roof set RF* formed by the g biggest elements of W. In the case in which almost all the matrix elements have the same value such that they can be equivalently inserted into GD as well into RF, these elements can be chosen arbitrarily to form the two sets.

Building the pairs $[l'_j, u'_j]$, $j=1,\ldots,k$, that validate (9) is a particular *matching problem*. In fact, let us introduce the following bipartite graph $B=(GD, RF, E)$, where the arc set E is defined as follows:

$$E=\{(u,v) : u\in GD,\ v\in RF,\ u \text{ and } v \text{ not belonging to the same row and } u\leqslant v\}.$$

Associated to each arc $(u,v)\in E$ there is its weight $\omega(u,v)=v-u$, which is by construction non-negative.

In the case in which the number of rows of W is $m\leqslant 3$, we have that $GD\cap RF\neq\emptyset$; it does not affect the matching properties since in B arcs among nodes representing the same matrix element do not exist.

Consider a *matching*, i.e., an arc set $M\subseteq E$ such that no pair of arcs are incident to the same node. Its bottleneck value (in the following we refer to it as *value*) is given by $V(M)=\min\{\omega(u,v) : (u,v)\in M\}$; that is the value of M coincides with the smallest weight among its elements. Each arc of M corresponds to a pair (u,v) which can be grouped in the same column j by ensuring that the resulting range δ_j will be:

$$v-u\leqslant\delta_j.$$

The matching property that no pair of arcs are incident with the same node guarantees that the pairs are independent of each other and can be assigned to different columns, thus validating (9).

Let us now consider the following *fixed cardinality bottleneck matching problem*: find a matching M^* of B such that $|M^*|=k$ and its value is

$$V(M^*)=\max\{V(M) : M\subseteq E \text{ and } |M|=k\}. \tag{10}$$

It is easy to prove that, given a value $\bar{\delta}$, if there exists a k-cardinality matching M whose value is $V(M)\geqslant\bar{\delta}$, then $\bar{\delta}$ is a lower bound for the optimal value z of the objective function. It is also easy to prove the converse: if every k-cardinality matching M of B is such that $V(M)\leqslant\bar{\delta}$, then $\bar{\delta}$ is an upper bound for z. In fact, any possible combination among ground and roof elements causes at least one permuted column in the range to be strictly less than $\bar{\delta}$, thus the minimum among the column ranges. Consequently, the optimal value z is given by $V(M^*)$.

For that, the problem $P24$ is equivalent of finding a k-cardinality matching M^* of maximum bottleneck value, as defined in (10), in the bipartite graph B. In fact, at the end we have at the same time k pairs defined by the matching and the value $V(M^*)$ of the objective function. To obtain W' it is sufficient to group each pair in the same column and to spread the k pairs in the k columns, through $2k$ exchanges at most.

The general problem of finding a maximum cardinality bottleneck matching in a bipartite graph $G=(O, D, E)$ has been widely studied (see, for a general review, [1] and [5]).

In particular, Punnen and Nair in [10] propose an algorithm based on the binary search on the set of all possible bottleneck values. In every such step a cardinality matching problem in a bipartite graph has to be solved. In order to obtain the time complexity of

$O(|E|^{0.5} \times (|O| + |D|)^{1.5})$, they use the algorithm of Alt et al. in [2] for solving the cardinality matching problem approximately. Then the cardinality of a maximum matching can be checked by growing only a small amount of augmenting paths.

It is not difficult to adapt the algorithm proposed in [2] in order to build a bottleneck matching "enough close" to a given cardinality, and not exceeding it, with the same complexity. Thus, the binary search proposed in [10] solves the k-cardinality matching problem and, consequently, problem $P24$. Since our bipartite graph B has $4k-2$ nodes and $O(k^2)$ arcs (and possible bottleneck values), the time complexity to solve problem $P24$ is $O(k^{2.5})$ (ordering the arcs according to their weights costs $O(k^2 \log k)$).

3.4. Reordering all the columns

In this subsection we will analyze the following problems:

- $P25$: $min(u, \Sigma)$,
- $P26$: $max(l, \Sigma)$,
- $P27$: $min(\delta, U)$,
- $P28$: $min(\delta, \Sigma)$.

For all four problems, as proved in the following, the permuted matrix W' is obtained by iteratively grouping in the same column the maximum row elements not yet grouped. This is equivalent to sorting each row separately, e.g. from the biggest to the smallest element. This task can be done in $O(n \log k)$ time and requires, in the worst case, $O(n)$ exchanges. Let us indicate in the following by W^* such a sorted matrix.

First, let us analyze problem $P25$. The objective function, to be minimized, is the sum of the column inner maxima (8):

$$\sum_{j=1}^{k} u_j = \sum_{j=1}^{k} \max\{w'_{ij} : i = 1, \ldots, m\}. \tag{11}$$

Let $V \in \Pi(W)$; by $u_j(V)$ we indicate the maximum element in column j of matrix V and by $z(V) = \sum_{j=1}^{k} u_j(V)$ its objective function value. A first result is:

Lemma 2. *The matrix V' obtained from V by grouping all the maximum row elements in the same column is such that $z(V') \leqslant z(V)$.*

Proof. Let $\bar{j}$ be the index of any column of V containing the maximum element of the whole matrix. V' is obtained by moving the maximum element of each row from its current position to column $\bar{j}$, if it is not already located there. As a consequence, the inner maximum $u_{\bar{j}}(V') = u_{\bar{j}}(V)$ since that column contains the global maximum element; while, for any other column j, $u_j(V') \leqslant u_j(V)$ holds since the elements entering column j are not greater than the corresponding leaving elements. The result follows. □

If we repeat the same grouping technique on the second maximum row elements, not considering the already grouped column $\bar{j}$, we obtain another matrix whose objective function value is not worse than $z(V)$.

Consider now a matrix V in which h columns $\{j_1, j_2, \ldots, j_h\}$, $0 < h < k$, contain the h biggest row elements, sorted in such a way that each element of column j_s is greater than or equal to the corresponding element of column j_{s+1}, for $s = 1, \ldots, h-1$. We call such a matrix *h-sorted*.

Lemma 3. *If V is h-sorted, matrix V' obtained from V by grouping in the same column the maximum row elements belonging to the remaining $k-h$ columns, is such that $z(V') \leqslant z(V)$.*

Proof. The result follows directly, by induction, from Lemma 2 and the subsequent observation. □

Lemma 3 proves the following theorem:

Theorem 3. *W^* is optimal for problem $P25$.*

Problem $P26$ is symmetric to $P25$ since it is based on the maximization of the inner minima; consequently, W^* is also a solution for $P26$.

In problem $P27$ we want to minimize the biggest inner range. The strategy for reducing the range of the column containing the biggest element of W is to group in that column all the maximum row elements. The observation also holds for the other columns, similarly to what is stated in Lemma 3, and again, matrix W^* is optimal for $P27$. Note that it is optimal for $P28$. Indeed, minimizing the sum of the inner ranges is obtained by iteratively grouping the maximum row elements in the same column.

4. NP-completeness results

We start the analysis of NP-hard cases from problem:

- $P29$: $min(\sigma, \Delta)$;

i.e., finding a k-partition $W^* \in \Pi(W)$ of minimum width:

$$\min\{\max\{\sigma(C_j) - \sigma(C_{j'}) : C_j, C_{j'} \in W', j, j' = 1, \ldots, k\} : W' \in \Pi(W)\}.$$

Next, we give two different reductions for the decision version of this problem proving it is NP-complete in the strong sense even if $m = 3$ (k arbitrary), and NP-complete in the ordinary sense even if $k = 2$ (m arbitrary). Moreover, we show that for $m = 2$ (k arbitrary) it is solvable in polynomial time.

First, let us consider the decision version of problem $P29$ which we denote with D1.

Problem D1. Let $W = [w_{ij}]$ be an integer matrix with m rows and k columns. Given an integer number H, does there exists $W' \in \Pi(W) : \max\{\sigma(C_j) - \sigma(C_{j'}) : C_j, C_{j'} \in W', j, j' = 1, \ldots, k\} \leqslant H$?

Theorem 4. *Problem* D1 *is NP-complete in the strong sense even if $m = 3$.*

Proof. We reduce the Numerical 3-Dimensional Matching Problem (N3-DM), known to be NP-complete in the strong sense [6], to D1.

Problem N3-DM

Instance: Disjoint sets T, X, and Y, each containing k elements, a size $s(a) \in Z^+$ for each element a, and a bound $B \in Z^+$.

Question: Can $T \cup X \cup Y$ be partitioned into k disjoint sets $C_1, C_2, \ldots, C_k$ such that each C_i contains exactly one element for each of T, X, and Y and such that, for $1 \leqslant i \leqslant k$, $\sum_{a \in C_i} s(a) = B$?

In order to reduce N3-DM to D1 we consider a k-partition problem with matrix $W = [w_{ij}]$ with 3 rows and k columns. To each $a \in T$, we associate an element w_{1j} with $w_{1j} = s(a)$, to each $a \in X$ an element $w_{2j} = s(a)$, and to each $a \in Y$ an element $w_{3j} = s(a)$. We choose $H = 0$ as the threshold for the corresponding decision problem.

If N3-DM has a solution, then there exists a partition $W' = \Pi(W)$ such that $\sigma(C_j) = \sigma(C_{j'})$ $j, j' = 1, \ldots, k$ which solves problem D1. If, on the other hand, problem D1 has a solution with $\max\{\sigma(C_j) - \sigma(C_{j'}) : C_j, C_{j'} \in W', j, j' = 1, \ldots, k\} = 0$ then for each C_j, $j = 1, \ldots, k$ it holds that $w_{1j} + w_{2j} + w_{3j} = B'$, and B' must be equal to B as $\sum_{ij} w_{ij} = kB$, then N3-DM has a solution. □

Theorem 5. *Problem* D1 *is NP-complete in the ordinary sense even if* $k = 2$.

Proof. We reduce Partition to D1.

Problem Partition

Instance: Finite set A and size $s(a) \in Z^+$ for each $a \in A$.

Question: Is there a subset A' of A such that $\sum_{a \in A'} s(a) = \sum_{a \in A - A'} s(a)$?

Note that Partition remains NP-complete even if the elements in A are ordered as $a_1, a_2, \ldots, a_n$ and we require that A' contains exactly one of a_{2l-1}, a_{2l} for $l = 1, \ldots, n/2$ [6].

Assuming the elements in A are ordered as above, for reducing Partition to D1 we consider the following instance of k-partition with $W = [w_{ij}]$ an integer matrix with m rows and 2 columns and with $w_{i1} = a_{2l-1}$, $w_{i2} = a_{2l}$, $i = 1, \ldots, m$ and $l = 1, \ldots, n/2$. We ask if a partition $W' \in \Pi(W)$ exists such that $\max\{\sigma(C_1) - \sigma(C_2) : C_1, C_2 \in W'\} = 0$.

If Partition has a solution, then there exists a k-partition $W' = \{C_1, C_2\}$ such that $\sigma(C_1) = \sigma(C_2)$, and thus with $\sigma(C_1) - \sigma(C_2) = 0$ which solves the decision problem D1.

Conversely, if problem D1 has a solution with $\sigma(C_1) - \sigma(C_2) = 0$ then $\sigma(C_1) = \sigma(C_2)$ from which it follows $\sum_{a \in A'} s(a) = \sum_{a \in A - A'} s(a)$. □

Now we analyze the following problems:

- $P30$: $min(\sigma, U)$,
- $P31$: $max(\sigma, L)$.

We can give reductions for problems $P30$ and $P31$ following Theorems 4 and 5. For this purpose the decision versions of problems $P30$ and $P31$ are given next:

Problem D2. Let W be an integer matrix with m rows and k columns. Given an integer number $H \in Z$, $\exists W' \in \Pi(W) : \max\{\sigma(C_j), C_j \in W'\} \leqslant H$?

Problem D3. Let W be an integer matrix with m rows and k columns. Given an integer number $H \in Z$, $\exists W' \in \Pi(W) : \min\{\sigma(C_j), C_j \in W'\} \leqslant H$?

Theorem 6. *Problems* D2 *and* D3 *are both NP-complete in the strong sense even if* $m = 3$ *and NP-complete in the ordinary sense even if* $k = 2$.

Proof. These proofs are similar to those of Theorems 4 and 5, and are omitted here. □

We conclude the analysis on NP-hard cases adding some final comments on instances with $m = 2$ and $k = n/2$.

The decision version of problems $P30$ and $P31$ can be answered in polynomial time. In fact, for any fixed value of threshold H we can construct a bipartite graph with n vertices and an edge between pairs of vertices if and only if the corresponding elements w_{1j} and $w_{2j'}$ are such that:

$$w_{1j} + w_{2j'} \leqslant H \quad \text{(problem D2, } P30\text{)},$$

$$w_{1j} + w_{2j'} \geqslant H \quad \text{(problem D3, } P31\text{)}.$$

The search for the minimum H for which a maximal (perfect) matching in this bipartite graph exists can be performed by the approach discussed in Section 3.3. In this specific case the bipartite graph has k nodes and k^2 edges, hence the proposed adaptation of the algorithm of Punnen and Nair would find the solution in time $O(k^{2.5})$. Since $k = O(n)$, the complexity is $O(n^{2.5})$.

As far as problem $P29$ is concerned in the particular case of $m = 2$ and $k = n/2$, to apply the same approach it would be required to solve even more matching problems. A simple way to tackle this is the following. Choose arbitrarily one couple w_{1t} and $w_{2t'}$ and consider the bipartite graph with $2k$ nodes and an edge between w_{1j} and $w_{2j'}$ if and only if:

$$w_{1t} + w_{2t'} - H/2 \leqslant w_{1j} + w_{2j'} \leqslant w_{1t} + w_{2t'} + H/2.$$

The existence of a perfect matching in such a graph implies the existence of a k-partition W' with $\max\{\sigma(C_j) - \sigma(C'_j)\} \leqslant H$. As the couple w_{1t} and $w_{2t'}$ has been chosen arbitrarily, in a first rough analysis we should consider all possible couples and hence solve $O(k^2) = O(n^2)$ maximal matching problems for each value of $H \in Q$ and verify if the value of the matching is k. The overall complexity of the optimization problem would then be bounded from above by $O(n^{4.5})$.

5. On the open problem

The last problem is

- $P32$: $min(\delta, \Delta)$.

This is the only one out of the 32 problems for which complexity remains open. We just discuss a graph theoretical interpretation which, however, has not yet led to a solution. It can be shown that the decision version of $P32$ is equivalent to finding a partition in cliques

of size k of a particular graph. While this latter problem is NP-complete on general graphs, to the best of our knowledge, no complexity results are known for the specific class which we get by reducing the decision version of $P32$.

First, let us consider the decision version of problem $P32$ which we denote with D4.

Problem D4. Let $W = [w_{ij}]$ be an integer matrix with m rows and k columns. Given an integer number H does there exist $W' \in \Pi(W)$: $\max\{\delta(C_j) - \delta(C_{j'}) : C_j, C_{j'} \in W', j, j' = 1, \ldots, k, \ j \neq j'\} \leqslant H$?

The graph problem can be obtained as follows. Assume we ask if there exists a k-partition satisfying Problem D4. For each w_{ij} build an interval $[w_{ij} - H/2, w_{ij} + H/2]$ on the real line. Define a graph with a vertex for each interval and an edge between vertices if and only the corresponding w_{ij} and $w_{i'j'}$ have $j \neq j'$ and the intervals overlap. Note that the first condition, i.e., $j \neq j'$, does not permit to have an edge even though the closed intervals do overlap, thus this graph is not an interval graph. Indeed, it is easy to verify that is not even a triangulated graph.

6. Applications

The topic covered in the paper offers solutions to many problems in parallel computing, files allocations, scheduling, and routing. In the following examples we exhibit some k-partition problems applied to lots scheduling and to routing on multigraphs.

Let us consider the following scheduling framework (see [3] for standard scheduling terminology). These are m production facilities. A set of k lots is to be executed in any order without interruption. Lots are of different size and hence a different processing time. Each lot is assigned to a facility. Suppose the production activity is organized on the basis of working periods, for instance on a daily basis, i.e., each facility processes one lot per day starting the execution at the beginning of the day. We are asked to find a schedule (i.e., the sequence of lot processing for each facility) which optimizes some performance criterion.

In such a scenario, the problem could be investigated with respect to different objective functions related to organizational requirements. We may be interested in balancing the daily workload among all m facilities, i.e., we want to find a schedule such that for each day the difference between the completion time on the last freed machine and the completion time of the first freed machine is minimized. Denote as w_{ij} the duration of the processing of lot j on facility i. A reasonable objective function for that problem could be to minimize $\sum_{j=1}^{k} (\max\{w_{ij} : i = 1, \ldots, m\} - \min\{w_{ij} : i = 1, \ldots, m\})$ which corresponds exactly to solving problem $P28$.

Another acceptable balancing criterion could be minimizing over all days the maximum difference between completion times of the same day (i.e., minimize $\max\{w_{ij} : i = 1, \ldots, m, \ j = 1, \ldots, k\} - \min\{w_{ij} : i = 1, \ldots, m, \ j = 1, \ldots, k\}$). Equivalently, we are looking for a k-partition with minimum (δ, U) (see problem $P27$). Both problems can be solved efficiently using algorithms described in Section 3.4.

Alternatively, we might be more concerned about the daily workload of the team of workers tending all the facilities which, for day j, can be measured as $\sum_{i=1}^{m} w_{ij}$. In this case, either we choose to minimize the maximum workload over all the days (minimize

$\max\{\sum_{i=1}^{m} w_{ij} : j = 1, \ldots, k\}$), or want to flatten the workload curve minimizing the difference between the maximum and minimum workload over all the days (minimize $\max\{\sum_{i=1}^{m} w_{ij} : j = 1, \ldots, k\} - \min\{\sum_{i=1}^{m} w_{ij} : j = 1, \ldots, k\}$), we have to face an NP-hard problem. In fact, the k-partition problems $P30$ and $P29$ modeling the above cases, have been proved to be NP-hard in the strong sense (see the Section 4).

Similar applications in problems related to assembly line balancing are discussed in [3].

Let us now examine examples of some k-partitions and routing problems in multigraphs. A graph can be called an *m-simple-multipath* if it is possible to have each couple of sequential nodes connected by exactly m edges of lengths $w_{ij}, i = 1, \ldots, m$. The routing problem consists in finding m simple arc-disjoint paths, each formed by k edges so that the paths are as uniform as possible [7,8]. One may wish to minimize, for example, the sum of the ranges between the longest and the shortest edge in the path obtaining a route with quite balanced edges length. Alternatively, one may want to determine the set of paths in which the difference between the longest and the shortest is minimum. It can be easily recognized that the first problem is $P23$ and the latter is problem $P29$.

Several other applications can be derived from real life combinatorial optimization problems.

7. Conclusions

In this work we defined and examined 32 uniform k-partition problems or, equivalently, 32 matrix permutation problems. They are characterized by the particular measure of "set uniformity" to be optimized. 21 of the studied problems can be solved by linear time algorithms, 7 require more complex algorithms but can still be solved in polynomial time, and 3 are proved to be NP-hard. The complexity of only one problem, namely $P32$, remains open.

Acknowledgements

The authors wish to thank anonymous referees whose valuable comments allowed us to improve the paper.

References

[1] R.K. Ahuja, T.L. Magnanti, J.B. Orlin, Network Flows, Prentice-Hall, Englewood Cliffs, NJ, 1993.
[2] H. Alt, N. Blum, K. Mehlhorn, M. Paul, Computing maximum cardinality matching in time $O(n^{1.5}\sqrt{m/\log n})$, Inform. Process. Lett. 37 (1991) 237–240.
[3] J. Blazewicz, K. Ecker, E. Pesch, G. Schmidt, J. Weglarz, Scheduling Computer and Manufacturing Processes, Springer, Berlin, 1996.
[4] M. Blum, R.W. Floyd, V.R. Pratt, R.L. Rivest, R.E. Tarjan, Time bounds for selection, J. Comput. System Sci. 7 (1973) 448–461.
[5] R.E. Burkard, Selected topics on assignment problems, Discrete Appl. Math. 123 (2002) 257–302.
[6] M.R. Garey, D.S. Johnson, Computers and Intractability: A Guide to the Theory of NP-Completeness, Freeman, New York, 1979.
[7] P. Hansen, Bicriterion path problems, in: Multiple Criteria Decision Making: Theory and Applications, Lecture Notes in Economics and Mathematical Systems, vol. 177, 1980, pp. 109–127.

[8] P. Hansen, G. Storchi, T. Vovor, Paths with minimum range and ratio of arc lengths, Discrete Appl. Math. 78 (1997) 1–3, 89–102.
[9] S. Martello, W.R. Pulleyblank, P. Toth, D. de Werra, Balanced optimization problems, Oper. Res. Lett. 3 (1984) 275–278.
[10] A.P. Punnen, K.P.K. Nair, Improved complexity bound for the maximum cardinality bottleneck bipartite matching problem, Discrete Appl. Math. 55 (1994) 91–93.