Università di Pisa

Dipartimento di Informatica

Technical Report: TR-01-07

On uniform k-partition problems

Paolo Dell'Olmo , Pierre Hansen , Stefano Pallottino , Giovanni Storchi

May 16, 2001

Address: Corso Italia 40, 56125 Pisa, Italy.

Tel: +39 050 2212700

Fax: +39 050 2212726

On uniform k-partition problems *

Paolo Dell'Olmo †, Pierre Hansen ‡, Stefano Pallottino §, Giovanni Storchi ¶

May 16, 2001

Abstract

In this work we study various uniform k-partition problems which consist in partitioning a collection of m sets, each of them of cardinality k, into k sets of cardinality m such that each of these sets contains exactly one element coming from every original set. The problems differ according to the particular measure of "set uniformity" to be optimized. Most of the studied problems are polynomial and the 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.

Key words: Partition, matrix permutation, algorithms, complexity.

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 $|C_j| = m, j = 1, \ldots, k$. In the following, we will 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 permutating 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 will present the 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

*Research partially supported by grants: FCAR of Québec, NSERC of Canada, MURST and INDAM-GNCS of Italy.

†Dipartimento di Statistica, Probabilità e Statistiche Applicate, Università di Roma "La Sapienza", P.le Aldo Moro 5, 00185 Roma, Italy, e-mail: dellolmo@pow2.sta.uniroma1.it

‡GERAD and École des Hautes Etudes Commerciales, Université de Montréal, 5255 av. Decelles, Montréal, Canada, H3T1V6, e-mail: pierreh@crt.umontreal.ca

§Dipartimento di Informatica, Università di Pisa, Corso Italia 40, 56125 Pisa, Italy, e-mail: pallo@di.unipi.it

¶Dipartimento di Statistica, Probabilità e Statistiche Applicate, Università di Roma "La Sapienza", P.le Aldo Moro 5, 00185 Roma, Italy, e-mail: giovanni.storchi@uniroma1.it

row i, we say that the columns of matrix $W' = \pi(W)$ are a *k-partition* of the rows of W, or, for the sake of brevity, W' is a *k-partition* of W.

Various measures can be adopted to evaluate the k-partition. These are based on both of an *inner measure* among the elements belonging to the same column and a *global measure* among the inner measures associated to the columns. In particular, as explained in the following, we adopt four different measures, namely the *minimum*, the *maximum*, the *difference between maximum and minimum*, or *range*, and the *sum*, or *value*. By combining these four measures at the inner and at the global levels, and by either maximizing or by 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 the NP-completeness proof for the intractable ones; while the classification of only one problem remains open. For convenience, we grouped the problems into classes according to their solution characteristics and computational complexity. Although the main contribution of the paper is rather theoretical, we also discuss different 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 performance criteria (or objective functions). In Section 3 we show which cases are "trivial", while Section 4 contains all the other polynomial cases. The NP-hard ones are presented in Section 5. In Section 6 we show an open problem. Finally, in Section 7 we discuss several applications of the k-partition problems and further researches 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 will denote the permutation of all m rows of W. Let $W' = \pi(W)$ be the *permuted matrix*; we will denote by $w'_{ij} = w_{i\pi_i(j)}$ the i-th element of column C_j of W' obtained by a permutation $\pi_i(W)$. Let us introduce more formally the four different *inner measures* of the k columns $\{C_1, \ldots, C_k\}$ of W'. Let

$$u_j = u(C_j) = \max\{w_{i\pi_i(j)} : i = 1, \ldots, m\} \tag{1}$$

and

$$l_j = l(C_j) = \min\{w_{i\pi_i(j)} :\} \tag{2}$$

be 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 by σ_j we indicate its *value*:

$$\sigma_j = \sigma(C_j) = \sum_{i=1}^{m} w_{i\pi_i(j)}. \tag{4}$$

Let us use the generic form f_j to indicate one of the four above mentioned inner measures for C_j, that is the j-th column of W'. Let us now examine the four *global measures* among the columns of matrix W'; more formally, we denote by:

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

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

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

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

the *maximum*, the *minimum*, the *range* and the *value* 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; in such a way, we obtain 32 different optimization problems.

To denote one of the 32 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 one.

For example, to maximize the range (7) among the columns, where the inner measure is the minimum (2), this will be denoted by $max(l, \Delta)$, that is to find the 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 swapping 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.

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	$k^{2.5} logk$	n	$n \log n$	n
σ	NP-hard	n	n	n

It should be noted that some of the above problems admit a trivial solution, as pointed out in the following Section. Nevertheless, for the sake of completeness in the classification of the k-partition problems, we also consider them.

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

3 The "trivial" problems

This class contains all the problems for which the objective function is a constant value for any $W' \in \Pi(W)$ and all the partitions (but just one case) are optimal; so, the input matrix W can be seen as the permuted matrix W' without any exchange operation. Let us consider the following problems:

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

For both problems the optimal value is given by $z = w_{max} = max\{w_{ij} : i = 1, \ldots, m, j = 1, \ldots, k\}$, i.e. a constant value which can be easily found through 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 given by $z = w_{min} = min\{w_{ij} : i = 1, \ldots, m, j = 1, \ldots, k\}$.

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:

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

Consider now the problem:

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

since we want to maximize the biggest range among 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 this case it is easy to prove that the biggest range is obtained either by w_{max} and the smallest one not belonging to the same row, or by the w_{min} and the biggest one not belonging to the same row (more precisely, by the greater among these two ranges). Also this case can be evaluated in $O(n)$ time and the permuted matrix W' is obtained from W through at most one exchange operation.

4 The classes of polynomial problems

Among the polynomial problems, we can distinguish three classes based on the exchanges necessary to obtain W':

1. reordering one or two columns;
2. spreading elements on the k columns;
3. reordering all the columns.

4.1 Partitions obtained by reordering one or two columns

This class is indeed subdivided into two subclasses that we present separately:

- the problems in which the reordered columns contain the maximum or minimum row elements,
- the problems in which the reordered columns contain elements belonging to a specific interval.

4.1.1 Grouping maximum and/or minimum row elements

The following three problems have as permuted matrix W' any matrix derived by 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 big as possible. Such a column is, indeed, the one containing the maximum element for each row. Determining the maximum of all the rows costs $O(n)$, and, with at most m exchanges, W is transformed into W'.

Analogously, for problem $P9$, we want to maximize the biggest 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 big as possible, that is a column with the maximum of row i as the i-th element, regardless of the other columns. The same permuted matrix solves problem $P10$, where the range (7) between the minimum 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, but they are based on the 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:

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

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

4.1.2 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).

Let us introduce some useful notation and definitions. A real interval $[a, b]$ is called *usable* for matrix W if there exists at least one element in each row of W belonging to that interval.

Note that, 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 so also an upper bound too of the optimal value z among all the 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 denoted in the following as 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 union of all values of the w's elements of W, $|T| = t \leq n$; 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 can be considered;

2. between two usable intervals $[a', b]$ and $[a'', b]$ such that $a' < a''$ only the latter can be considered; the same consideration can be made between $[a, b']$ and $[a, b'']$ with $b' < b''$ where only the first can be considered;

3. if an interval $[a, b]$ is not usable, by denoting with I the set of rows not having elements belonging to $[a, b]$, the minimum width usable interval $[a, b']$ having a as the inferior extreme 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 inferior extreme $\geq a$ exist;

4. if $[a, b]$ is usable, the minimum width usable interval $[a', b]$ having b as superior extreme is such that:

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

5. given an usable interval $[a, b]$ such that $[a', b]$ is not usable for any $a < a' \leq b$, the next inferior extreme 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 superior extreme b' is found and, through observation 4 the inferior extreme a' is detected such that $[a', b']$ is a usable interval of the sequence, and of minimal width 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 $t \leq n$.

Proof: Each time a new interval is built, its inferior extreme is strictly greater than the inferior extreme of the previous interval. Since the number of different w's values is t, this thesis follows. □

Let $Q = \{[a_1, b_1], [a_2, b_2], \ldots, [a_q, b_q]\}$ with $q \leq t$, 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 $[a_{h+1}, b_{h+1}]$.

It is easy to prove the following Theorem

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 $a_{h+1} = a' \geq a'' > a_h$; thus, also the second property 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. $\Box$

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

Procedure *Interval*:
  begin {initialization}
    Sort rows $R_1, \ldots, R_m$; compute $\widehat{w}$ and $\bar{w}$;
    $\bar{a} := w_{min}; \bar{b} := b_1 := \widehat{w}; \bar{\delta} := \bar{b} - \bar{a}; h := 1;$ *EmptyHeap*;
    for $i := 1$ **to** m **do**
      begin {initializing the heap}
        while $j(i) < k$ **and** $w_{ij(i)+1} \leq b_1$ **do** $j(i) := j(i) + 1$;
        AddHeap(i)
      end;
    repeat {main loop}
      MinHeap(i); $a_h := w_{ij(i)}$;
      if $b_h - a_h < \bar{\delta}$ **then begin** $\bar{a} := a_h; \bar{b} := b_h; \bar{\delta} := \bar{b} - \bar{a}$ **end**;
      if $a_h < \bar{w}$ **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** $w_{ij(i)+1} \leq b_h$ **do** $j(i) := j(i) + 1$;
              UpdateHeap(i)
            end
        end
    until $a_h = \bar{w}$;
    return $\{\bar{a}, \bar{b}, \bar{\delta}\}$
  end.

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 $i, i = 1, \ldots, m$, 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 superior extreme b_h:

$$j(i) = argmax\{w_{ij} : j = 1, \ldots, k, w_{ij} \leq b_h\}.$$

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 inferior extreme 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.

In the initialization, the two values $\widehat{w}$ 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 accordingly to the initial value b_1, and the first heap is built.

In the main loop, the minimum key gives the inferior extreme 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 (that is $w_{ij(i)} = a_{h-1}$), the new key is selected and the superior extreme 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 inferior extreme 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 the problem $P15$ is obtained.

Let us analyze now 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 $q \leq t \leq n$ 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 superior extreme 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 same time complexity $O(n \log m)$ has the last loop **for ... do**; 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 Theorem.

Theorem 2: Algorithm *Interval* is correct and runs in $O(n \log n)$ time in the 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 $O(logn)$, this thesis follows. □

As far as problem $P16$ is concerned, we recall that its objective function to be maximized is $max\{\delta_j : j = 1, \dots, k\} - min\{\delta_j : j = 1, \dots, k\}$. 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, $P16$ maintains the same time complexity as $P15$.

4.2 Partitions obtained by spreading elements on k columns

We divide this class into two different subclasses:

- the problems in which k elements are selected and are spread into k different columns,
- a problem in which k pairs of elements are selected and then reordered and spread into k different columns.

4.2.1 Spreading k elements on k columns

Examine 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 k-th biggest element among n can be solved in $O(n)$ time [3]; 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 k-th biggest value. The same for problem $P18$ in which the sum of the maximum column values is the objective function to maximize. Looking at 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 k-th 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.

4.2.2 Spreading k pairs of elements on k columns

In this subsection we will analyze the following problem:

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

Let us suppose to know a lower bound lb of the optimum value of the objective function; that is, 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

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

Given a column h and a value u'_h belonging to it, in order to validate (9), one can select the minimum value element among the ones that can be paired with u'_h. Note that the $k-1$ w's which belong to the same row of u'_h can not 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 $j \neq h$.

Since, in the worst case, it is possible that the smallest $2k-2$ elements of matrix W can not be paired with a given element u'_h, 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 state also the *roof set* RF formed by the g biggest elements of W. In the following we consider that $GD \cap RF \neq \emptyset$, that is that the number of rows is $m > 3$.

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

$$E(lb) = \{(i,j) : i \in GD, j \in RF, j - i \geq lb, i \text{ and } j \text{ not belonging to the same row}\}.$$

Let us build now a matching of cardinality k, i.e. a subset $M(lb)$ of $E(lb)$ such that no pair of arcs are incident in the same node. Each arc of $E(lb)$ corresponds to a pair $[l'_j, u'_j]$ which can be grouped in the same column j by ensuring that the resulting range δ_j will be:

$$u'_j - l'_j \leq \delta_j.$$

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

It is easy to prove that, given a value $\bar{\delta}$, if the corresponding bipartite graph $B(\bar{\delta})$ contains a matching $M(\bar{\delta})$ of cardinality k, then $\bar{\delta}$ is a lower bound for the optimal value z of the objective function. It is also easy to prove the opposite: if the maximum cardinality matching $M(\bar{\delta})$ of the bipartite graph $B(\bar{\delta})$ is less than k, 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.

One algorithmic approach to problem $P24$ is to define a set of possible values of the objective function, order them and apply a *dichotomous search* by solving a matching problem at each iteration.

In order to analyse complexity we need to start from the cardinality of $E(lb)$. Being $|GD| = |RF| = g = 2k - 1$ it follows that $|E(lb)| \leq g^2$ and thus $|E(lb)| = O(k^2)$. Ordering $E(lb)$ costs $O(k^2 logk)$.

During the dichotomous search on $E(lb)$, for the given middle value $\bar{v}$ of the alive portion of $E(lb)$, we test whether $B(\bar{v})$ contains a matching $B(\bar{v})$ of cardinality k; if so, we take the remaining portion of $E(lb)$ as the one above the current value $\bar{v}$ and if not we take the other part. When the portion of v reduces to a single value $\hat{v}$ for which a k-cardinality matching $M(\hat{v})$ in $B(\hat{v})$ exists, we have at the same time k pairs defined by the matching and the value $\hat{v}$ 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.

Theorem 3: The problem $max(\delta, L)$ can be solved optimally in $O(k^{2.5} logk)$

Proof The optimality has already been proved in the description of the procedure. For complexity, we shown that computing the initial ordered sequence of values $E(lb)$ costs $O(k^2 \log k)$ time. The dichotomous search requires to solve $O(\log k^2) \rightarrow O(\log k)$ matching problems.

The matching algorithm runs in $O(k^{2.5})$ (see [1]). Thus the overall complexity of the procedure is $O(k^{2.5} logk)$. $\square$

4.3 Partitions obtained by 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 this 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 elements. 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):

$$\sum_{j=1}^{k} u_j = \sum_{j=1}^{k} max\{w_{i\pi_i(j)} : i = 1, \dots, m\}. \tag{10}$$

Let V be any matrix obtained from W through a sequence of permutations between elements of the same rows. 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 $z(V') \leq z(V)$.

Proof: Let $\bar{j}$ be the index of the 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, $u_j(V') \leq u_j(V)$ holds since the elements entering column j are not greater than the corresponding leaving elements. This thesis 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 $\{j_1, j_2, \dots, j_h\}$, $0 < h < k$, 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 $s = 1, \dots, h-1$. 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 $z(V') \leq z(V)$.

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

Lemma 3 proves the following Theorem:

Theorem 4: 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 $P27$. Notice that it is optimal for $P28$ too, since in this problem we minimize the sum of the inner ranges.

5 NP-completeness results

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

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

that is finding a k-partition $W^* \in \Pi(W)$ of minimum width:

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

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), 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 there exists $W' \in \Pi(W) : max\{\sigma(C_j) - \sigma(C_{j'}) : C_j, C_{j'} \in W', j, j' = 1, \ldots, k,\ j \neq j'\} = H$?

Theorem 5: 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 [4], 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, …,C_k such that each C_i contains exactly one element for each of T, X, and Y and such that, for $1 \leq i \leq k$, $\sum_{a \in C_i} s(a) = B$?

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 the problem D1. If, on the other hand, problem D1 has a solution with $max\{\sigma(C_j) - \sigma(C_{j'}) : C_j, C_{j'} \in W', j, j' = 1, \ldots, k, j \neq j'\} = 0$ 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 6: 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 $l = 1, \ldots, n/2$ [4]).

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.

Vice versa, 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 5 and 6. 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 $H \in Z$

$$\exists W' \in \Pi(W) : max\{\sigma(C_j), C_j \in W'\} = H \text{ ?}$$

Problem D3. Let W be an integer matrix with m rows and k columns. Given an integer number $H \in Z$

$$\exists W' \in \Pi(W) : min\{\sigma(C_j), C_j \in W'\} = H \text{ ?}$$

Theorem 7: Problem 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: (see proofs of Theorems 5, 6).

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

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 $w_{2j'}$ with $j \neq j'$ are such that:

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

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

The search for the existence of a maximal (perfect) matching on this bipartite graph would answer the question in time $O(k^{2.5})$. The corresponding optimization problems can then be solved in $O(k^{2.5}\log H)$ by applying dichotomous search in the interval $[0, H = max\{w_{1j} + w_{2i}, j = 1,\dots,k, j' = 1,\dots k,\ \ j \neq j'\}]$ to find the minimum H for which a perfect matching exists. Since $H = O(w_{max})$ and $k = O(n)$, the complexity is $O(n^{2.5}\log w_{max})$.

As far as problem $P29$ is concerned, to apply the same approach we 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:

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

The existence of a perfect matching on such a graph implies the existence of a k-partition W' with $max\{\sigma(C_j) - \sigma(C'_j)\} \leq H$. 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 and verify if the value of the matching is k. The overall complexity of the optimization problem would then be bounded from above by $O(n^{4.5}\log w_{max})$.

6 On the open problem

Consider the problem

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

This is the only one out of the 32 problems for which complexity remains open. We just discuss a graph theoretical interpretation which, however, does not bring to any result. 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 by get 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 there exists $W' \in \Pi(W) : max\{\delta(C_j) - \delta(C_{j'}) : C_j, C_{j'} \in W', j, j' = 1,\dots,k,\ j \neq j'\} \leq H$?

The graph problem can be obtained as follows. Assume we ask if there exists a k-partition satifying 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 one 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 permits to have an edge even though the closed intervals do overlap, thus this graph is not an interval graph. Indeed, it is not even a triangulated graph. One can easily verify that a chordless cycle of length four can be obtained by the minimal example of a 2×2 matrix with the first row formed by 1's and the second row formed by 2's, with $H = 1$. We could not for see any help from such an approach which we report just to stimulate further research.

7 Applications

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

Let us consider the following scheduling framework (see [2] for standard scheduling terminology). Let there be m production facilities. To each facility i is assigned a set of k lots of different size and hence a different processing duration, to be executed in any order. Lots must be processed without interruption. 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 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 in relation 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 $\sum_{j=1}^{k}(max\ w_{ij} - min\ w_{ij}), i = 1, \ldots, m$ which corresponds exactly to solving problem $P29$.

Another acceptable balancing criterion could be minimizing over all days the maximum difference between completion times of the same day (i.e. minimize $(max\ w_{ij} - min\ w_{ij}), j = 1, \ldots, k, i = 1, \ldots, m)$. 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 4.3.

Differently, 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 latter 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 $(max\ \sum_{i=1}^{m} w_{ij}$ - $min_j \sum_{i=1}^{m} w_{ij}), j = 1, \ldots, k$), 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 5).

Similar applications in problems related to assembly line balancing are discussed in [2].

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 paths, each formed by k edges so that the paths are as uniform as possible. 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. Differently, 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.

8 Conclusions

In this work we defined and examined 32 uniform k-partition problems or, equivalently, 32 matrix permutation problems. These 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 are still be solved in polynomial time, and 3 are proved to be NP-hard. The complexity of only one problem, namely $P32$, remains open.

References

[1] R. K. Ahuja, T. L. Magnanti, J. B. Orlin. *Network Flows*, Prentice Hall, 1993.

[2] J. Blazewicz, K. Ecker, E. Pesch, G. Schmidt, J. Weglarz. *Scheduling Computer and Manufacturing Processes*, Springer, 1996.

[3] M. Blum, R. W. Floyd, V. R. Pratt, R. L. Rivest and R. E. Tarjan, Time bounds for selection , *Journal Computing and System Science*,7, 448-461, 1973.

[4] M. R. Garey and D. S. Johnson. *Computers and intractability: a guide to the theory of NP-completeness*, Freeman, 1979.

[5] P. Hansen. Bicriterion Path Problems. in Multiple Criteria Decision Making: Theory and Applications, *Lecture Notes in Economic and Mathematical Systems*, 177, 109-127, 1980.

[6] P. Hansen, G. Storchi and T. Vovor. Paths with Minimum Range and Ratio of Arc Lengths, *Discrete Applied Mathematics*, (78) 1-3, 89-102, 1997.