Most uniform path partitioning and its use in image processing

Mario Lucertini
Department of Electrical Engineering and Centro Matematico Vito Volterra, University of Roma "Tor Vergata", via O. Raimondo, 00173 Roma, Italy

Yehoshua Perl
Institute for Integrated Systems, Computer and Information Science Department, New Jersey Institute of Technology, Newark, NJ 07102, USA

Bruno Simeone
Department of Statistics, University of Roma "La Sapienza", Piazza A. Moro 5, 00185 Roma, Italy

Received 15 July 1990
Revised 1 March 1991

Abstract

Let \( Q \) be a vertex-weighted path with \( n \) vertices. For any pair \((L,U)\) can one find a partition of \( Q \) into \( (a \) given number \( p \) of) subpaths, such that the total weight of every subpath lies between \( L \) and \( U \)? We present linear-time algorithms for the partitioning problem for given \((L,U)\) and an \( O(n^2 \log n) \) algorithm, relying on the above procedures, for finding a partition that minimizes the difference between the largest and the smallest weight of a subpath (most uniform partitioning). Our approach combines a preprocessing procedure, which detects "obstructions", if any, via a sequence of vertex compressions; and a greedy procedure, which actually finds the desired partition. Path partitioning can be a useful tool in facing image degradation. In fact whenever a picture is taken or converted from one form to another, the resulting image can be affected by different types and degrees of degradation; if we have no informations on the actual degradation process that has taken place on the image (or if it is too difficult or costly to find such informations), the only way for image enhancement consists in increasing contrast and reducing noise by suitable modifications of the grey level of pixels. Finding the optimal grey scale transformation which leads to this enhancement can be formulated as the problem of partitioning into connected components a path with vertices corresponding to grey levels and vertex weights equal to the number of occurrences of the corresponding tone in the image, so that the sum of the weights of the vertices in each component is "as constant as possible". In addition to image processing, this problem has applications in paging, clustering and the design of communication networks.

Correspondence to: Professor B. Simeone, Department of Statistics, University of Roma “La Sapienza”, Piazzale Aldo Moro 5, 00185 Roma, Italy.
1. Introduction

In image processing, a digital image is stored in the computer memory as a two-dimensional array of pixels, ranging, say, from 1 (white) to 256 (black). Since in many applications a subdivision of the grey scale into 16 different tones of grey is desirable, the 256 tones are segmented into 16 “bands” formed by contiguous tones of grey. It has been empirically observed that the best optical contrast is achieved when the total number of pixels belonging to each band is “as constant as possible”. Thus the problem of finding the best image can be formulated as the problem of partitioning into 16 connected components a path with 256 vertices, whose weights are equal to the number of occurrences of the corresponding tone in the image, so that the sum of the weights of the vertices in each component is “as constant as possible” [2]. The image processing problems with this objective function are generally referred to as “histogram equalization” problems [8]. In a more general framework path partitioning can be a useful tool in facing image degradation. In fact whenever a picture is taken or converted from one form to another, the resulting image can be affected by different types and degrees of degradation, so it could be difficult to distinguish details. If we have no informations on the actual degradation process that has taken place on the image (or if it is too difficult or costly to find such informations), the only way for image enhancement consists in increasing contrast and reducing noise by suitable modifications of the grey level of pixels. The grey scale transformation which leads to this enhancement consists in a suitable mapping from the original set of grey levels (domain) to a new (smaller) set of grey levels (codomain). To obtain an enhancement we choose the set of grey levels of the codomain sufficiently different from each other and we preserve the order of grey levels of the original image, i.e., we assign segments of the grey scale of the domain to the grey levels of the codomain. The mapping is chosen so that:

(i) There will be enough pixels in each component to point out clearly regions with different grey levels.

(ii) The (small) grey level differences among pixels in the same component (that can depend on noise effects such as salt and pepper or haze) will be eliminated. This technique achieves the double goal of using the full range of the display to point out regions with different grey level and smoothing the small fluctuations of grey level from point to point within the same region of the image [15].

If we denote by $L$ and $U$ the minimum and the maximum number of pixels in each component, respectively, we can say that the smaller is the value $(U-L)$ the better is the contrast and the quality of the image. The partitioning problems with this objective function are generally referred to as “most uniform partitioning” (MUP).

In addition to image processing, this problem has applications in paging, clustering and the design of communication networks.

MUP is not the only optimization criterium which can be used to achieve the desired results. In previous papers we have considered different optimization criteria
for tree partitioning. In [12] a shifting algorithm is presented for the max-min partitioning problem where the weight of the minimum component is maximized. In [4] another shifting algorithm is given for the min-max partitioning problem where the weight of the maximum component is minimized. A more efficient implementation of this algorithm appears in [14]. Generalizations of those two algorithms for general weighting functions appear in [3]. Actually the most desired partitioning criterion for many applications is the uniform partitioning where each of the components has the same weight. But such a partition does not always exist. One actually realizes that the max-min and the min-max partitioning are approximations for the uniform partition. To see this note that in case a uniform partition exists, it is obtainable both as a max-min and as a min-max partition. In [5] we consider another approximation for the uniform partition, the equi-partition of a tree where we minimize the sum of the differences between the weights of the components and the weight of the average component. However it is shown that the equi-partitioning problem for a general tree is NP-hard. Algorithms are presented for special families of trees. A better approximation for the uniform partition is the most uniform partition where we minimize the difference between the largest and the smallest weight of the components. Obviously, if this difference is zero, then the uniform partition is obtained.

In Fig. 1, we demonstrate that the most uniform partition is a better approximation to the uniform partition than both the max-min partition and the min-max partition. In this example the heaviest component of the most uniform partition has a weight slightly greater than the heaviest component in the min-max partition. At the same time the lightest component of the most uniform partition has a weight slightly smaller than the lightest component in the max-min partition.

The problem of finding a most uniform partition is difficult, since in order to minimize the difference of the weights of the heaviest and lightest components one needs to take simultaneously into account both the weight of the heaviest component and the weight of the lightest component. The experience is that whenever one tries to optimize on two parameters at the same time the problem gets more difficult.

| Partition of a path of \( n = 10 \) weighted vertices into \( p = 7 \) subpaths (let \( L \) and \( U \) be the total weight of the lightest and the heaviest subpath respectively). |
|-----------------|-----|-----|-----|-----|-----|-----|-----|-----|
| The path:       | 10  | 49  | 2   | 7   | 50  | 3   | 50  | 10  |
| Max-min partition: \((L = 10, U = 57)\) | 10  | 49  | 2   | 7   | 50  | 3   | 50  | 10  | 10    |
| Min-max partition: \((L = 5, U = 50)\) | 10  | 49  | 2   | 7   | 50  | 3   | 50  | 10  | 10    |
| Most uniform partition: \((L = 9, U = 53)\) | 10  | 49  | 2   | 7   | 50  | 3   | 50  | 10  | 10    |

Fig. 1.
For example in [1] we deal with two weighting functions at the same time. It is shown that minimizing the weight subject to a constraint on the size of the components is NP-hard. On the other hand for the problem of minimizing the weight while having a given constraint on the height of the components a very complicated polynomial shifting algorithm is presented. In [13] we consider tree partitioning into a minimum number of components under given constraints on both the size of each component and the capacity of the connections of each component. The problem is shown to be NP-hard and a pseudo-polynomial algorithm is given.

In order to tackle the most uniform partitioning problem we define a related problem, the \((L, U)\)-partitioning problem, where the weight of each component must lie between the values \(L\) and \(U\) (where \(L \leq U\)). This problem is also interesting by itself and has applications in paging and clustering. The related problem of partitioning a tree into the minimum number of components such that the weight of each component is at most \(U\) was solved in linear time in [10]. A linear algorithm for another related problem, i.e., partitioning a tree into the maximum number of components such that the weight of each component is at most \(L\) is given in [12]. However, when the components have to satisfy both bounds \(L\) and \(U\), the problem is much more complicated.

Thus, on the ground of the image processing application described above, we decided as a first step to analyze the problem for the special case of a path. To our surprise even this restricted problem, typical (in slightly different forms) of many image processing applications, is far from being trivial, besides being interesting by itself. In this paper linear algorithms are presented for a set of related problems:

1. \((L, U)\)-partitioning (from now on \(\Pi(L, U)\)).
2. Minimum \((L, U)\)-partitioning (from now on \(\Pi_{\text{min}}(L, U)\)), i.e., \((L, U)\)-partitioning into the minimum number of components.
3. Maximum \((L, U)\)-partitioning (from now on \(\Pi_{\text{max}}(L, U)\)), i.e., \((L, U)\)-partitioning into the maximum number of components.
4. \((L, U)\)-partitioning into \(p\) components (from now on \(\Pi_p(L, U)\)), i.e., \((L, U)\)-partitioning into a specified number \(p\) of components.

Moreover two polynomial algorithms, both based on \((L, U)\)-partitioning, are presented for the most uniform path partitioning problem (from now on MUP).

To state formally the problem, let us introduce some notations. Let \(Q = (V, E)\) be a path with \(n\) vertices. A nonnegative weight \(w_i\) is associated with every vertex \(i \in V\). A \(p\)-partition \(\pi\) of \(Q\) is any partition of \(Q\) into \(p\) subpaths (components) \(C_1, C_2, \ldots, C_p\) which can be obtained by the deletion of \(p - 1\) edges of \(Q\) (\(1 \leq p \leq n\)). The weight \(W(C_i)\) of a component \(C_i\) is the sum of the weights of its vertices.

The problem introduced above can now be written as:

\[\Pi(L, U)\]: Given the path \(Q(V, E)\), the set of weights \(\{w_i\}\), find a partition \(\pi = \{C_1, C_2, \ldots, C_p\}\) such that:

\[L \leq W(C_i) \leq U, \quad \text{for } i = 1, 2, \ldots, p.\]
Most uniform path partitioning

\[ \Pi_{\min}(L, U) \]: Given the path \( Q(V, E) \), the set of weights \( \{w_i\} \), find a partition \( \pi = \{C_1, C_2, \ldots, C_p\} \) such that:

(i) \( L \leq W(C_i) \leq U \), for \( i = 1, 2, \ldots, p \);

(ii) \( p \) is minimum.

\[ \Pi_{\max}(L, U) \]: Given the path \( Q(V, E) \), the set of weights \( \{w_i\} \), find a partition \( \pi = \{C_1, C_2, \ldots, C_p\} \) such that:

(i) \( L \leq W(C_i) \leq U \), for \( i = 1, 2, \ldots, p \);

(ii) \( p \) is maximum.

\[ \Pi_p(L, U) \]: Given the path \( Q(V, E) \), the set of weights \( \{w_i\} \) and a positive integer \( p \), find a partition \( \pi = \{C_1, C_2, \ldots, C_p\} \) such that:

\[ L \leq W(C_i) \leq U, \quad \text{for } i = 1, 2, \ldots, p. \]

MUP: Given the path \( Q(V, E) \), the set of weights \( \{w_i\} \) and a positive integer \( p \), find a partition \( \pi = \{C_1, C_2, \ldots, C_p\} \) such that:

(i) \( L \leq W(C_i) \leq U \), for \( i = 1, 2, \ldots, p \);

(ii) \( z - U - L \) is minimum.

For general graphs the problem of finding a most uniform partition into \( p \) connected components is NP-hard, even for the case of unit weights of the vertices. The reduction is from the problem of partitioning a graph into paths of length 2 [7].

Here is the outline of the paper. In Section 2 we motivate and informally describe the basic algorithm for \( \Pi(L, U) \). We believe that the heuristic process of algorithmic development is often more important than its final product, namely the algorithm itself; we felt that, in this case, we had a good opportunity to provide the reader with a real-life and instructive example of development of an algorithm by trial and error. In Section 3 we introduce parallelogram posets and prove some basic results about them. In our opinion, parallelogram posets offer a simple and crystal-clear view of the algorithms; the preprocessing procedures described in the paper are nothing but paths along the poset, and the right-to-left procedure is the rightmost path. In Section 4 we formally describe the algorithms for \( \Pi(L, U) \) and give correctness and complexity proofs. Moreover we present linear-time algorithms for \( \Pi_p(L, U) \). Finally, in Section 5, we describe two algorithms for MUP, relying on the above procedures.

2. Towards linear time algorithms

Perhaps the most common strategy for the solution of an optimal path partitioning problem is dynamic programming. Basically, the problem is reduced to finding a shortest path in a suitable acyclic network (see e.g. [5]). By a similar approach,
one could reduce the recognition of \((L, U)\)-partitionable paths to finding a path in a certain acyclic network \(N\).

Actually, given the weighted path \((Q, w)\), the network \(N\) is built as follows. The vertex set of \(N\) is \(\{0, 1, \ldots, n\}\), and there is an edge \((i, j)\) \((0 \leq i < j \leq n)\) whenever
\[
L \leq w_{i+1} + \cdots + w_j \leq U.
\]
Clearly, \(Q\) is \((L, U)\)-partitionable if and only if there exists a path from 0 to \(n\) in \(N\).

Sometimes one wants to check the existence of an \((L, U)\)-partition with a specified number \(p\) of components. To this purpose, a layered network \(M\) is introduced. The vertices of \(M\) are the pairs:

- \((0, 0)\),
- \((k, i)\), \(k=1, \ldots, p-1; k \leq i \leq n-p+k;\)
- \((p, n)\)

where the pair \((k, i)\) represents the fact that the vertex \(i\) is the last vertex of component \(k\) (notice that in any \(p\)-partition vertex \(i\) can be the last vertex of component \(k\) only if \(k \leq i \leq n-p+k\)). For all \(k=1, \ldots, p-2\), there is an edge from \((k, i)\) to \((k+1, j)\) iff \(i < j\) and \(L \leq w_{i+1} + \cdots + w_j \leq U\). In addition, one introduces all edges from \((0, 0)\) to \((1, i)\), \(i=1, \ldots, n-p+1;\) and all edges from \((p-1, i)\) to \((p, n)\), \(i=p-1, \ldots, n-1\). There is a one-to-one correspondence between the \((L, U)\)-partitions into \(p\) components of \(Q\) and the paths from \((0, 0)\) to \((p, n)\) in \(M\). For example, if \(n=7\), \(p=3\), \(L=5\) and \(U=9\), the 3-partition \(\{2 \ 3 \ 4 \ 5 \ 2 \ 5\}\) corresponds to the path \((0, 0), (1, 2), (2, 5), (3, 7)\). One can detect the existence of a path in \(N\) or in \(M\) in time proportional to the number of edges in the network. The problem is that the number of edges is \(O(n^2)\) in \(N\) and \(O(n^2p)\) in \(M\). Thus, this approach does not lead to linear-time algorithms.

In many applications, in particular real-time applications like image processing with images given by a robot vision system, the complexity of the above path finding procedure is too high. This is especially true since our main purpose is an efficient algorithm for MUP. As we shall see in Section 5 many problems \(\Pi_p(L, U)\) must be solved for solving MUP. Thus we look for a more efficient algorithm for \(\Pi_p(L, U)\).

A greedy algorithm. For this purpose we shall consider a greedy approach. We shall start by discussing a straightforward greedy technique. Let us first consider \(\Pi_{\min}(L, U)\) where a greedy approach looks natural. To obtain a minimum partition such that the weight of every component is at most \(U\) we perform a linear scanning of the path, accumulating the weight \(w\) of a component and assigning a cut to the edge leading to the vertex whose weight, when added to the current \(w\), would cause \(w\) to exceed \(U\). However, we have to be careful about satisfying the other requirement that the weight of each component is at least \(L\). We shall try this approach on a small example; this will also prove the difficulty of the problem and develop some intuitive insight into its structure.

In Fig. 2(a) we see that after assigning three cuts we are stuck: we cannot assign
Most uniform path partitioning

- Fig. 2. Straightforward (9,13)-scanning.

the fourth cut and still have a (9,13)-partition since the fourth component is either too light or too heavy. Hence we must move some previously assigned cuts. Thus we assign the fourth cut to the starred position and start shifting the previous cuts to the left in order to satisfy the requirements. We adopt here a stingy approach, shifting each cut as little as possible. As soon as the cut C(3) is shifted, we are able to resume the greedy procedure assigning the cuts C(5) and C(6), and obtaining the partition in Fig. 2(b). But now we are stuck again, since the seventh component is either too light or too heavy. So we apply again the shifting procedure to rearrange the previous cuts. Actually only C(6) is shifted one position to the left. Now we can assign C(7) to obtain the partition in Fig. 2(c). At this point we are almost done, but there is another problem. The last component is too heavy but cannot be partitioned into two components. Thus we have to apply again the shifting procedure to rearrange the previous cuts. In this case all seven cuts are shifted and then the eighth cut can be assigned to obtain the (9,13)-partition into nine components in Fig. 2(d). Note that this partition is, by coincidence, a maximum partition as well as a minimum partition, so it is the only feasible (9,13)-partition for this path (i.e., a (9,13)-partition into \( k \neq 9 \) components of this path is not possible). In this example the greedy approach led to the desired solution. However the example indicates a problem with regard to the complexity of the algorithm: each time we add a cut we might have to shift all previous cuts. This implies that the complexity of the procedure would not be linear. In view of those difficulties it seems that a deeper analysis of the problem is necessary for the design of an algorithm.

Preprocessing. Let us review the above described procedure. Suppose the scanning gets stuck after the cut C(i) is assigned to the position j between the jth and the \((j+1)\)th vertices. The reason why we cannot assign the next cut C(i+1) is that up to some position, say \( m \), the accumulated weight of the current component is lighter than \( L \) and if we add the next weight \( w_m \) the component gets heavier than \( U \). Thus we can conclude that no \((L,U)\)-partition may have a cut at the position \( j \) or that the position \( j \) is forbidden. Note that this conclusion is independent of the assignment of other cuts on the left or on the right of the \( j \)th position. In the example
of Fig. 2 every time the scanning gets stuck this is due to attempting to assign a cut to a forbidden position. We can avoid the scanning from getting stuck by not allowing to assign a cut to a forbidden position. Or equivalently by combining two vertices on the left and on the right of the forbidden position into one vertex whose weight is the sum of the weights of the two vertices. We call this operation compression. We shall show later that if a weight larger than $U$ is obtained during this process then there is no feasible solution to our problem. Thus the idea is to preprocess the path in order to check whether each position is forbidden or not, and perform the appropriate compressions for the forbidden positions.

We can perform this preprocessing in an efficient way by scanning the path from right to left. Actually a position, say $m$, which was not recognized to be forbidden, might be recognized as such after a compression has occurred on the right of the position. Thus a new compression must be performed. If we process the path from right to left, rather than from left to right, we are able to identify any position which becomes forbidden due to another forbidden position, immediately in the first stage rather in later stages. Hence right-to-left preprocessing, unlike the left-to-right one, requires a single scanning of the path. In Fig. 3 we demonstrate the right-to-left preprocessing on the same example of Fig. 2. Note that we still accumulate weights on the right of a position as before, only the order of processing the positions is changed.

In the preprocessing procedure all forbidden positions are found and all required compressions are done in one scanning. As we show later, after this single stage there are no more forbidden positions. In this example, the preprocessing happens to compress the vertices into vertices with weights between $L$ and $U$. Thus in this

\[
\begin{array}{cccccccccccc}
11 & 2 & 4 & 3 & 2 & 5 & 6 & 2 & 4 & 3 & 7 & 2 & 1 & 1 & 1 & 1 & 1 & 4 & 3 & 2 & 2 & 3 & 5 & 6 & 11 \\
\end{array}
\]

Fig. 3. Right-to-left preprocessing for a $(9,13)$-partition.
Most uniform path partitioning

In Section 3 it will be proved that any sequence of compressions leads to the same compressed path, but the right-to-left preprocessing is the only one that requires one scanning of the path.

Of course one could reverse the role of right and left, so as to get a symmetric procedure (possibly leading to a different set of forbidden edges). We shall expand this issue in Section 4.

Using a queue. The complexity of preprocessing is not clear since we cannot know how many weights we have to add until the sum exceeds $L$. If all weights are integers then we add each time no more weights than $L$. A trivial lower bound for the complexity is then $O(mL)$. However, it is possible to reorganize the preprocessing such that the complexity is linear. The idea is to use the weight accumulated for checking position $m$ in order to compute the weight necessary for checking the position $m - 1$. To compute the accumulated weight for position $m - 1$, we add to the weight accumulated for position $m$ the weight $w_m$ and then we subtract the last, the second last, ..., weight from the accumulated weight until it gets into the range between $L$ and $U$. In case the accumulated weight does not eventually get into the prescribed range, then the position $m - 1$ is forbidden and a compression of vertices $m - 1$ and $m$ is performed.

We present now the preprocessing procedure. The index POS points to the vertex on the right of the position being currently checked. The weights of the vertices in the component to the right of the position POS are inserted into the queue QUEUE with sum of the weights WC. Compressions are done while entering the queue where a new weight is added to the last inserted weight. The weights are checked for being larger than $U$ (in this case no $(L, U)$-partition exists) while being deleted from the queue. The weights are then transferred to a second array AW filling the locations $i + 1, \ldots, n$. Note that due to the compressions there are less weights in AW than in the original array of weights.

We demonstrate the right-to-left implementation of preprocessing in Fig. 4. In this example some compression occurs, but after the preprocessing not all weights are larger or equal to $L$ as in the previous example.

A formal proof that the above preprocessing procedure requires only $O(n)$ time will be given in Section 4 on the ground of the analysis of the compression process given in Section 3. Assume that the path has been preprocessed, and that during the preprocessing no vertex got a weight larger than $U$. At this point it is immediate to obtain an $(L, U)$-partition: just execute the above described greedy algorithm. This time such procedure can never get stuck, because all forbidden positions have been eliminated by preprocessing.

$(L, U)$-partitions into a given number $p$ of components. Let $r$ and $s$ be the minimum and the maximum number of components, respectively, in an $(L, U)$-partition. The number $r$ can be computed by the greedy algorithm described above. Similarly, $s$ can be computed by a “stingy” procedure, in which we assign a cut as soon as the
Fig. 4. Linear implementation of right-to-left preprocessing for a (9,13)-partition.

accumulated weight of a component became greater than or equal to $L$. Figures 5(a) and (b) show the outcome of the greedy and stingy procedures, respectively.

Given an integer $p$, $r<p<s$, how do we find an $(L,U)$-partition into $p$ components? Although in this case we cannot use the greedy or the stingy procedure directly, it turns out that we can combine the minimum and the maximum partitions to obtain an $(L,U)$-partition for every $p$ such that $r<p<s$. 
Most uniform path partitioning

1 2 1 | 1 2 1 2 | 1 2 1 2 | 1 2 1 2 | 1 2 1 2 | 1 2 3
(a) Minimum (3,5)-partition into six components

1 2 | 1 2 1 2 | 1 2 1 2 | 1 2 1 2 | 1 2 3
(b) Maximum (3,5)-partition into nine components

1 2 | 1 2 1 2 | 1 2 1 2 | 1 2 1 2 | 1 2 3
(c) (3,5)-partition into seven components

1 2 | 1 2 1 2 | 1 2 1 2 | 1 2 1 2 | 1 2 3
(d) (3,5)-partition into eight components

Fig. 5. \((3,5)\)-partition of a path.

For the path of Fig. 5 one has \(r = 6\) and \(s = 9\). Partitions into six, nine, seven and eight components are exhibited in Fig. 5(a), (b), (c), (d), respectively. It is interesting to notice that both in the 7-partition and in the 8-partition all cuts are either cuts of the minimum partition or cuts of the maximum partition. Furthermore, both in the 7- and in the 8-partition all cuts of the minimum partition are on the right of all cuts of the maximum partition. We shall see in Section 4 that these properties hold in general and that they lead to linear algorithms for finding \((L, U)\)-partitions for all \(p\) such that \(r < p < s\).

In the following the algorithms for: Preprocessing, Maximum partition, Minimum partition and Partition into \(p\) components, are presented in a Pascal-like format. Formal proofs of the correctness of the algorithms will be presented in Sections 3 and 4.

Procedure Preprocessing.

Input: \(\{w_1, \ldots, w_n\}, L, U\).

Output: \(\{x_1, \ldots, x_{n-1}\} \cdot [x_i = 1\) if vertices \(i\) and \(i + 1\) collapse, \(x_i = 0\) otherwise\]

\(x:=0; WC:=0; i:=n; j:=n; w_0:=U; \) QUEUE := Ø;

while \((j > 0)\) do
  begin
    \(\text{while } (WC \leq U) \text{ do}
      \begin{align*}
        & \text{begin}
        & \text{while } (WC \leq U) \text{ do}
        & \begin{align*}
          & \text{begin}
          & \text{QUEUE} := \text{QUEUE} \cup \{i\};
          & WC := WC + w_i;
          & i := i - 1;
        & \end{align*}
        & \text{end}
        & \text{if } (WC - w_{i+1} < L) \text{ then } x_{i+1} := 1;
        & \text{QUEUE} := \text{QUEUE} - \{j\};
        & WC := WC - w_j;
        & j := j - 1;
      & \end{align*}
    & \end{align*}
  & \end{align*}
\)
Procedure Minimum partition.
Input: \( \{w_1, \ldots, w_n\}, L, U \) [\( w_i \leq U \) for all \( i \), else no solution exists]
Output: \( \{x_1, \ldots, x_{n-1}\} \). [\( x_i = 1 \) if there is a cut between vertices \( i \) and \( i+1 \), \( x_i = 0 \) otherwise]

\[
x := 0; \quad WC := 0; \quad i := 1; \quad w_{n+1} := U;
\]
while \( (i < n) \) do
  begin
    while \( (WC \leq U) \) do
      begin
        WC := WC + \( w_i \); \( i := i + 1 \);
      end
    \( x_{i-1} := 1 \);
    WC := \( w_i \);
  end

Procedure Maximum partition.
Input: \( \{w_1, \ldots, w_n\}, L, U \) [\( w_i \leq U \) for all \( i \), else no solution exists]
Output: \( \{y_1, \ldots, y_{n-1}\} \). [\( y_i = 1 \) if there is a cut between vertices \( i \) and \( i+1 \), \( y_i = 0 \) otherwise]

\[
y := 0; \quad WC := 0; \quad i := 1; \quad w_{n+1} := U;
\]
while \( (i < n) \) do
  begin
    while \( (WC < L) \) do
      begin
        WC := WC + \( w_i \); \( i := i + 1 \);
      end
    \( y_i := 1 \);
    WC := 0;
  end

Procedure Partition into \( p \) components.
Input: \( \{w_1, \ldots, w_n\}, L, U, p \) (with \( r < p < s \)), a Minimum partition, a Maximum partition.
Output: A \( p \)-partition.

\[
j := 1;
\]
while (the \( j \)th maximum cut is on the left of the \( (j+r+2-p) \)th minimum cut) do
  begin
    take the \( j \)th maximum cut as \( j \)th \( p \)-cut; \( j := j + 1 \);
  end
while \( (j = p) \) do
  begin
    take the \( (j+r+2-p) \)th minimum cut as \( j \)th \( p \)-cut; \( j := j + 1 \);
  end
3. Parallelogram posets and compression lattices

Given the weighted path $Q = \{1, \ldots, n\}$, let us add two dummy vertices 0 and $n+1$, both having weight $U$. Clearly, $Q$ is $(L, U)$-partitionable if and only if $Q' = \{0,1,\ldots,n,n+1\}$ is such. Thus, without loss of generality, we may assume that the path $Q$ under consideration satisfies the conditions $w_1 = w_n = U$. In this case we shall call $Q$ a standard path. This assumption is introduced for technical reasons to simplify the notation.

Given a partition $\pi = \{C_1, \ldots, C_p\}$ of the path $Q$, the left border of a component $C_k = \{r+1, \ldots, s\}$ of $\pi$ is the edge $(r, r+1)$; the right border of $C_k$ is the edge $(s, s+1)$; an edge is said to be a border if it is either a left border or a right border of a component $C_k$. By definition, $C_1$ has no left border and $C_p$ has no right border. An edge $e = (i, i+1)$ is forbidden if there is no $(L, U)$-partition where $e$ is a border. In particular all edges are forbidden if $Q$ is not $(L, U)$-partitionable.

A sufficient condition for $e$ to be forbidden is that there exists an index $k$, $i+1 \leq k \leq n$, such that (throughout this paper, we follow the convention that if $r > s$ then the integer interval $[r, s]$ is empty and any sum $\sum_{i=r,\ldots,s} a_i$ is zero):

$$\sum_{h=i+1,\ldots,k-1} w_h < L \quad \text{and} \quad \sum_{h=i+1,\ldots,k} w_h > U. \quad (1)$$

In this case $e$ will be called (left-)illegal; otherwise, $e$ is called (left-)legal; in the latter case there exists a $k > i$ such that $L \leq \sum_{h=i+1,\ldots,k} w_h \leq U$.

One similarly defines the corresponding right-concepts. Whenever we do not specify “left-” or “right-”, we always mean “left-”.

If $e = (i, i+1)$ is any illegal edge, the compression of vertices $i$ and $i+1$ is the operation which consists in collapsing $i$ and $i+1$ into a single vertex and in assigning to the new vertex the weight $w_i + w_{i+1}$. We shall say that a (weighted) path $Q''$ is a compression of the (weighted) path $Q'$ (and we write $Q'' \leq Q'$) if $Q''$ can be obtained from $Q'$ through a finite sequence of compression operations.

Consider the set $\mathcal{Q}$ of all compressions of $Q$. Then the relation $\leq$ is a partial order in $\mathcal{Q}$. It turns out that the poset $\mathcal{Q}$ has very strong properties: in fact we shall see that it is a semimodular lattice. This will bear some consequences on the correctness proof of the preprocessing algorithm. Actually we are going to distil some key properties of $\mathcal{Q}$ and turn them into axioms for a certain class of posets; and then we shall prove in general that every poset in this class is a semimodular lattice. Such lattice need not be a boolean lattice; however, we prove that it is “locally” boolean in a well-defined sense.

Our starting point is the following fundamental property of compressions.

**Lemma 1.** Assume that edges $e_i = (i, i+1)$ and $e_j = (j, j+1)$, $i < j$, are both illegal in the path $Q$. Then:

(a) $e_j$ remains illegal in the path $Q'$ obtained by compressing the vertices $i$ and $i+1$;
(b) $e_i$ remains illegal in the path $Q''$ obtained by compressing the vertices $j$ and $j+1$. 

\[ \text{Lemma 1. Assume that edges } e_i = (i, i+1) \text{ and } e_j = (j, j+1), \text{ } i < j, \text{ are both illegal in the path } Q. \text{ Then:} \]

\[ \begin{align*}
& \text{(a) } e_j \text{ remains illegal in the path } Q' \text{ obtained by compressing the vertices } i \text{ and } i+1; \\
& \text{(b) } e_i \text{ remains illegal in the path } Q'' \text{ obtained by compressing the vertices } j \text{ and } j+1.
\end{align*} \]
Proof. (a) is trivial since vertices \( i \) and \( i+1 \) are on the left of \( e_j \). (b) Since \( e_i \) is illegal there is an index \( k \) such that (1) are verified. The node weights of \( Q^* \) are given by:

\[
w'(h) = \begin{cases} 
    w(h), & \text{if } i + 1 \leq h < j, \\
    w(j) + w(j+1), & \text{if } h = j, \\
    w(h+1), & \text{if } j < h < n.
\end{cases}
\]

There is always an index \( l \) such that \( \sum_{h=i+1,\ldots,l-1} w'(h) < L \) and \( \sum_{h=i+1,\ldots,l} w'(h) > U \): namely choose \( l = k \) when \( k \leq j \) and \( l = k - 1 \) when \( k > j \). Hence the edge \( i \) is illegal. \( \square \)

Now, let \((\mathcal{P}, \leq)\) be any finite poset with a unit \( \top \), and let \( \delta = \{1, 2, \ldots, m\} \) be a finite set of colors. Let \( H = (\mathcal{P}, E) \) be the Hasse diagram of \((\mathcal{P}, \leq)\). A texture of \( H \) is a coloration of \( E \) (i.e., a map from \( E \) to \( \mathcal{P} \)) such that:

(i) For every \( x \in \mathcal{P} \), all edges going into \( x \) have different colors.

(ii) For every \( x \in \mathcal{P} \), all edges going out of \( x \) have different colors.

(iii) For every \( x \in \mathcal{P} \), and for every path \( \mu \) from \( \top \) to \( x \), all edges of \( \mu \) have different colors.

Notice that (iii) is equivalent to:

(iii') If \( q \) is an arbitrary path, all edges of \( q \) have different colors.

A textured poset \( \mathcal{P} \) is said to have the parallelogram property if, whenever \( a, b, c \in \mathcal{P} \) are such that \( a \) is a common predecessor of \( b \) and \( c \) and the edges \((a, b)\) and \((a, c)\) have the colors \( i \) and \( j \), respectively, then there exists a unique \( d \in \mathcal{P} \) with the following two properties:

(1) \( d \) is a common successor of \( b \) and \( c \);

(2) edge \((b, d)\) has the color \( j \) and edge \((c, d)\) has the color \( i \).

The 4-element configuration of Fig. 6(b) will be called a parallelogram and denoted by \( \Pi_{ij}(a, b, c, d) \). The motivation for introducing the class of textured posets with the parallelogram property (briefly, parallelogram posets) is that the poset \( \mathcal{Q} \) of all compressions of a path \( Q \) belongs to this class. The unit of \( \mathcal{Q} \) is \( Q \) itself; also notice that each edge \( u \) of the Hasse diagram \( H \) of \( \mathcal{Q} \) corresponds to a compression.
operation contracting a certain edge \( e = e(u) \) of \( Q \). The map \( u \mapsto e(u) \) is easily seen to be a texture. Finally, the parallelogram property holds by Lemma 1.

We now discuss some theoretical properties of parallelogram posets.

**Lemma 2.** If two elements of a parallelogram poset have a common predecessor, then they must have a unique common successor.

**Proof.** Let \( b, c \) be two elements having a common predecessor \( a \) and let the colors of edges \((a, b)\) and \((a, c)\) be \( i \) and \( j \), respectively. Then there exists a parallelogram \( \Pi_{ij}(a, b, c, d) \) for a unique \( d \in \mathcal{P} \). Assume that there is another common successor \( x \) of \( b \) and \( c \). The color \( k \) of edge \((c, x)\) must be different from \( i \) and \( j \) by axioms (i) and (ii). By the parallelogram property, there must be a parallelogram \( \Pi_{ij}(c, d, x, y) \) for some \( y \in \mathcal{P} \), with \( y \neq a, b, c, d, x \). But then in the path \( abxy \) the two edges \((a, b)\) and \((x, y)\) carry the same color, contradicting axiom (iii'). Thus \( x = d \).  

**Lemma 3.** Assume that \( a, b, c, d \in \mathcal{P} \) are such that:
- \((a, b)\) is an edge with color \( i \);
- \((a, c)\) and \((b, d)\) are edges with color \( j \).
Then \((c, d)\) is an edge with color \( i \).

**Proof.** There exists a parallelogram \( \Pi_{ij}(a, b, c, x) \) for some \( x \in \mathcal{P} \); one must have \( x = d \), for otherwise both edges \((b, d)\) and \((b, x)\) would carry color \( j \), contradicting axiom (ii).  

**Lemma 4.** Let \( a, b, c \in \mathcal{P} \) be such that:
- \((a, b)\) is an edge with color \( i \);
- there is a path from \( a \) to \( c \), let it be \( \beta \), which does not contain \( b \).
Then:
1. if some edge of \( \beta \) has the color \( i \), one must have \( b \geq c \);
2. if no edge of \( \beta \) has the color \( i \), there exists a successor \( d \) of \( c \) such that edge \((c, d)\) has the color \( i \) and \( b \geq d \).

**Proof.** Let \( \beta = x_1, \ldots, x_s \), with \( a = x_1 \), \( c = x_s \), and let \( j_h \) be the color of edge \((x_h, x_{h+1})\); set \( b = y_1 \) and let \( k \) be the smallest index such that edge \((x_k, x_{k+1})\) carries color \( i \). If \( k = 2 \), then (by Lemma 3) there must be an edge \((y_1, x_2)\) and (1) holds. Otherwise, there must be a parallelogram \( \Pi_{ij}(x_1, y_1, x_2, y_2) \) with \( y_2 \neq x_2 \) for all \( i \). If \( k = 3 \), then there exists an edge \((y_2, x_3)\) (by Lemma 3) and (1) holds. Otherwise there is a parallelogram \( \Pi_{ij}(x_2, y_2, x_3, y_3) \), and so on. In this way one can build a path \( y_1, y_2, \ldots \) with the property that all edges \((x_n, y_n)\) carry the same color \( i \). Eventually, there must be an edge from \( y_{k-1} \) to \( x_{k+1} \) (by Lemma 3) and hence (1) holds. When no edge of \( \beta \) has color \( i \), we can continue the above process until a successor \( y_s \) of \( x_s \) is eventually produced such that \((x_s, y_s)\) has the color \( i \). Thus (2) holds.
Theorem 5. Any parallelogram poset $\mathcal{P}$ is a lattice.

Proof. Since $\mathcal{P}$ is finite and has a unit, it will be enough to show that $\mathcal{P}$ is a meet semilattice (see [9, Section 1.3, Lemma 14]). So, let $x$ and $y$ be any two elements of $\mathcal{P}$. We want to prove the existence of a meet or greatest lower bound (glb) of $x$ and $y$. Since $\mathcal{P}$ has a unit, for every $z \in \mathcal{P}$ there is some path from $\top$ to $z$. Let $d(z)$ (depth of $z$) be the shortest length of any such path. Our proof proceeds by induction on $m = d(x) + d(y)$. The statement is trivial when $m = 1$ and is an immediate consequence of the parallelogram property when $m = 2$. So, assume that two elements have a meet whenever the sum of their depths is smaller than $m$, and let $x, y \in \mathcal{P}$ be such that $d(x) + d(y) = m \geq 3$ (without loss of generality we may assume that $d(x) \geq 2$). Let $\mu$ be a shortest path from $\top$ to $x$. Let $z$ be the predecessor of $x$ along $\mu$ ($z \neq \top$). By the inductive hypothesis, $y$ and $z$ have a meet $t = y \wedge z$. If $t \leq x$, then $t \leq x, y$ and any predecessor of $x, y$ is also a predecessor of $z$ and hence of $t$; thus $t$ is a glb of $x$ and $y$. If $t \leq x$ does not hold, then let $\mu$ be any path from $z$ to $t$. Now $x$ cannot belong to $\mu$. Let $i$ be the color of edge $(z, x)$. No edge of $\mu$ can carry color $i$, for otherwise one would have $t \leq x$ by Lemma 4(1). Hence, by Lemma 4(2), $t$ must have a successor $u$ such that edge $(t, u)$ has the color $i$ and $u \leq x$. Then $u$ is a common lower bound of $x$ and $y$. We claim that $u$ is a glb of $x$ and $y$. In fact, for every $w \leq x, y$, there exists a path $\mu'$ from $t$ to $w$. If no edge of $\mu'$ bears color $i$, then $w$ has a successor $u$ which bears color $i$, moreover there exists a path $\mu''$ from $x$ to $w$. The concatenation of $(z, x), \mu', (w, u)$ is a path in which the two edges $(z, x)$ and $(w, u)$ carry the same color $i$. But this contradicts axiom (iii'). Therefore some edge of $\mu'$ bears color $i$ and by Lemma 4(1) one has $w \leq u$. Thus $u$ is a glb of $x$ and $y$. $\square$

Theorem 6. Any parallelogram poset $\mathcal{P}$ is a lower semimodular lattice.

Proof. If two elements $b$ and $c$ of $\mathcal{P}$ have a common predecessor $a$ then, by Lemma 2, they also have a common successor $d$. It is easy to see that $a = b \vee c$ and $d = b \wedge c$. Hence $\mathcal{P}$ is lower semimodular. $\square$

Since the lattice $\mathcal{P}$ is semimodular, the Jordan–Hölder condition must hold:

If $x, y \in \mathcal{P}$ are such that $x \geq y$, then any two paths from $x$ to $y$ have the same length.

Actually, a stronger property holds, as shown by the next result. If $Q$ is a path, let us denote by $\partial(Q)$ the set of colors carried by the edges in $Q$.

Theorem 7. If $x \geq y$, then for any two paths $U$ and $V$ from $x$ to $y$ one has $\partial(U) = \partial(V)$ (but of course the order of colors along $U$ and along $V$ is different).

Proof. It will be enough to prove the theorem when $x = \top$; the general case easily
follows. The proof is by induction on $d(y)$ (depth of $y$). When $d(y) = 2$ the thesis follows from Lemma 2 and the parallelogram property. So, assume that the theorem is true for all $z$ such that $d(z) \leq k - 1$ and consider an arbitrary $y$ such that $d(y) = k$. If $U$ and $V$ have the last edge in common, then the thesis follows at once from the inductive hypothesis. Thus assume, without loss of generality, that the last edge of $U$ is $(u, y)$ and that the last edge of $V$ is $(v, y)$, where $u \neq v$ and $d(u) = d(v) = k - 1$, and $i$ and $j$ the corresponding colors. As $P$ is semimodular $w = u \lor v$ is a common predecessor of $u, v$; let $\mu$ be any path from 1 to $w$. By the inductive hypothesis, $\partial(U - (u, v)) = \partial(\mu) \cup \{j\}$, $\partial(V - (v, y)) = \partial(\mu) \cup \{i\}$ and therefore $\partial(U) = \partial(\mu) \cup \{i\} \cup \{j\} = \partial(V)$.  

Let $B_n$ be the canonical boolean lattice whose elements are all the subsets of \{1, ..., n\}, partially ordered by inclusion.

In view of Theorem 7, one can define a map $g : \mathcal{P} \to B_n$ by $x \mapsto g(x) = \text{set of colors of the edges of any path from } x \text{ to } x$.

**Lemma 8.** The map $g$ is injective.

**Proof.** We shall prove the following statement $T$: \text{"Let } z \in \mathcal{P}, \text{ let } X \text{ be a path from } z \text{ to some } x \text{ and } Y \text{ a path from } z \text{ to some } y \neq x; \text{ then } \partial(X) = \partial(Y)". Clearly $T$ implies the thesis. $T$ is true when both $\partial(X) = \partial(Y)$ have cardinality 1. Suppose that $T$ is false. Choose a counterexample in which the cardinality of $\partial(X) = \partial(Y)$ is as small as possible (but of course $\geq 2$). Let $w$ be the successor of $z$ along the path $X$ and let $i$ be the color of edge $(z, w)$. By Lemma 4, since one of the edges of $Y$ carries the color $i$ there is a path $W$ from $w$ to $y$. Denote by $X'$ the subpath of $X$ from $w$ to $x$. One has $\partial(X') = \partial(X) - \{i\}$. Hence $\partial(X') = \partial(W)$. Thus we have found a counterexample to $T$ such that $|\partial(X')| = |\partial(W)| < |\partial(X)|$, contradicting the minimality assumption. Therefore $T$ holds.  

A parallelogram poset does not have to be a boolean lattice: for example the parallelogram poset of Fig. 7 is not a boolean lattice.

![Fig. 7.](image-url)
Definition. A finite lattice $L$ is said to be locally boolean if, for every $x \in L$, the sublattice generated by the successors of $x$ is boolean and hence isomorphic with $B_k$, where $k$ is the number of such successors.

For example, the parallelogram poset of Fig. 7 is seen to be locally boolean. In fact, this property holds for arbitrary parallelogram posets, as shown by Theorem 10 below.

Lemma 9. Let $L$ be the sublattice generated by the successors of $x$. Let $1, 2, \ldots, k$ be the colors of the edges going out of $x$. Then $L = \{ y: y \leq x \text{ and } \{1, \ldots, k\} \supseteq g(y)\}$.

Proof. Let $M = \{ y: y \leq x \text{ and } \{1, \ldots, k\} \supseteq g(y)\}$; $M$ is a parallelogram poset and hence a lattice. Obviously $M$ contains all successors of $x$, so it is enough to prove that every $y$ in $M$ belongs also to $L$. We use induction on the relative depth of $y$, i.e., the length of any path from $x$ to $y$. The statement is true for relative depth 1. Suppose it holds for relative depth $k-1$ and let $y$ be of relative depth $k$. Let $z$ be the predecessor of $y$ along a path from $x$ to $y$, and $w$ the predecessor of $z$. Let the color of edge $(z, y)$ be $i$. By Lemma 4, the set of colors carried by the edges going out of $w$ is $\{1, \ldots, k\} - g(w)$. But $i \not\in g(w)$, else axiom (iii) would be contradicted. Therefore there must be a successor $u$ of $w$ such that edge $(w, u)$ has the color $i$. As both $u$ and $z$ belong to $L$, also $y$ belongs to $L$.

Theorem 10. Every parallelogram poset $\mathcal{P}$ is a locally boolean lattice.

Proof. Let $x \in \mathcal{P}$, and let $S$ be the sublattice generated by the $k$ successors $x_1, \ldots, x_k$ of $x$. For simplicity let the color of edge $(x, x_i)$ be $i$ ($i = 1, \ldots, k$). By Lemma 9, $S = \{ y \in \mathcal{P}: y \leq x, \{1, \ldots, k\} \supseteq g(y)\}$. From Lemma 8, the map $h$ from $S$ to $B_k$ defined by $h(y) = g(y)$ for $y$ in $S$ is injective. By Lemma 4, for every $y$ in $S$ and for every color $i \in \{1, \ldots, k\} - g(y)$, there is an edge going out of $y$ and carrying the color $i$; it follows that for every $D \in B_k$ there exists a $z \in S$ such that $h(z) = D$; therefore $h$ is surjective. Clearly if $y, z \in S$ and $y \leq z$ then $h(z) \supseteq h(y)$. Thus $h$ is a lattice homomorphism.

So far the color set $\delta$ was not required to have any particular structure, i.e., $\delta$ was just an arbitrary finite set. Now we assume that $\delta$ is a linearly ordered set. We shall number the colors from 1 to $|\delta|$ in such a way that color $j$ follows color $i$ in the linear order if and only if $i < j$. Denote by $f(z)$ the greatest color of an edge going out of $z$. We shall require that the given linear order satisfies the axiom "For every edge $(x, y)$ of $H$, one has $f(x) \geq f(y)$". Again, the introduction of the axiom is motivated by the easy to prove fact that the lattice of compressions does satisfy such axiom, the linear order being in this case the left-to-right order of the edges along the path $Q$. For every element $x \in \mathcal{P}$, we shall assume that the edges going out of $x$ are sorted in increasing color order. We visualize this by imagining that the edges going out...
of $x$ are arranged from left to right so that the $k$th edge from the left is the one carrying the $k$th smallest color. Thus the meaning of terms such as "the rightmost path from $x$ to $y$" is evident.

**Definition.** A path $e_1e_2\ldots e_q$ is said to be decreasing if $\text{color}(e_1) > \text{color}(e_2) > \cdots > \text{color}(e_q)$.

**Proposition 11.** There exists one and only one decreasing path from $1$ to $0$ in $H$. Actually, this path is the rightmost path from $1$ to $0$.

**Proof.** Let $\pi$ be the rightmost path from $1$ to $0$. Obviously $\pi$ is decreasing. By Theorem 7, the colors of the edges along any other path $\pi'$ from $1$ to $0$ are a non-identical permutation of the colors of the edges along $\pi$. Hence $\pi'$ cannot be a decreasing path. □

**Remark.** For every $x$ in $\mathcal{X}$, let $g(x)$ be the set of edges to be contracted in order to obtain $x$. Professor Schrader pointed out to us that the set $\mathcal{F} = \{g(x): x \in \mathcal{X}\}$ is closed under union. This implies that $(E, \mathcal{F})$ is an antimatroid. Using some theoretical results on general antimatroids given in [6], one could prove that $\mathcal{D}$ is a lower semi-modular lattice. However, we have preferred to introduce parallelogram posets because of their simple and intuitive geometry.

### 4. Theoretical properties of $(L, U)$-partitions

In what follows $Q$ is a given vertex-weighted path, $L$ and $U$ are given and $\mathcal{D}$ is the lattice of compressions of $Q$ with respect to $L$ and $U$.

**Definition.** A path is said to be stable if it does not have any illegal edge: that is, if no compression can take place.

**Lemma 12.** There exists one and only one stable compression $Q^\circ$ of $Q$, and $Q^\circ$ is the zero of $\mathcal{D}$.

**Proof.** Since $\mathcal{D}$ is a finite lattice, it must have a unique zero $Q^\circ$. Clearly the zero is stable and must be a compression of any other element of $\mathcal{D}$. □

**Lemma 13.** Let $Q'$ be any compression of $Q$. Then $Q$ is $(L, U)$-partitionable if and only if $Q'$ is such.

**Proof.** Obvious. □

**Theorem 14.** The right-to-left preprocessing procedure outputs $Q^\circ$. 
Proof. By Lemma 12, it will be enough to show that the right-to-left procedure outputs a stable compression of $Q$. But the check for illegality in the preprocessing and in the partitioning algorithm are the same. Thus if an edge is illegal at the end of the preprocessing it must have been illegal at the moment of being checked during the preprocessing. \[ \square \]

**Definition.** A *scan-and-cut* procedure for a path $Q$ is any procedure that assigns cuts to some edges of $Q$ in such a way that, having assigned a cut to an edge $(h, h+1)$, it assigns the next cut to an edge $(k, k+1)$, where $k > h$ satisfies the condition:

$$L \leq \sum_{i=h+1, \ldots, k} w_i \leq U$$

(we make the convention that initially $h = 0$).

The procedure halts when $k = n - 1$ (we keep assuming that the weight $w_n$ of the last vertex is equal to $U$); or when there is no $k$ satisfying the above condition (2). In the latter case we say that the procedure "gets stuck".

Two extreme examples of scan-and-cut procedures are the *stingy procedure*, which at every step selects the smallest index $k$ satisfying (2); and the *greedy procedure*, which selects the largest such index.

**Lemma 15.** If $Q$ is an arbitrary path, two necessary conditions for $Q$ to be $(L, U)$-partitionable are:

1. the total weight of the vertices of $Q$ is at least $L$;
2. the weight of every vertex of $Q$ is at most $U$.

If, in addition, $Q$ is stable, then the two conditions are also sufficient. Moreover, any scan-and-cut procedure outputs an $(L, U)$-partition of $Q$.

**Proof.** The necessity of (1) and (2) is trivial. Assume that $Q$ is stable, and consider an arbitrary scan-and-cut procedure. Because of (1) and (2), the procedure can indeed assign the first cut. Since there are no illegal edges, the procedure can assign also the remaining cuts without ever getting stuck. Hence the scan-and-cut procedure outputs an $(L, U)$-partition of $Q$. \[ \square \]

**Theorem 16.** Let $Q$ be a standard path (i.e., $w_1 = w_n = U$). The following conditions are equivalent:

1. $Q$ is $(L, U)$-partitionable.
2. No compression of $Q$ has a vertex with weight strictly larger than $U$.
3. No vertex of $Q^\circ$ has weight strictly larger than $U$.
4. The weight of the first vertex of $Q^\circ$ is at most $U$.

**Proof.** (1) $\Rightarrow$ (2) follows from Lemma 13 and from the necessity part of Lemma 15. (2) $\Rightarrow$ (3) obvious.
Most uniform path partitioning

(3) = (1) by Lemma 12, \( Q^* \) is stable; by the sufficiency part of Lemma 15, \( Q^* \) is \((L, U)\)-partitionable; by Lemma 12, the path \( Q \) is also \((L, U)\) partitionable.

(3) = (4) obvious.

(4) = (3) suppose that \( Q^* \) has a node \( i \) with weight larger than \( U \); since (4) holds, one cannot have \( i > 1 \) either, because in this case edge \((i-1, i)\) would be illegal, against Lemma 12.

At this point, let us examine some consequences of the above results. Lemma 12 says that if, starting from \( Q \), we perform an arbitrary sequence of compressions, stopping only when no further compression is possible, then the resulting path \( Q^* \) is always the same, independent of the order of the compressions. In particular, according to Theorem 14, one (efficient) way to obtain such a path is to use the right-to-left preprocessing procedure. If during the execution of this procedure one detects a compression of \( Q \) having a vertex with weight larger than \( U \), then \( Q \) is not \((L, U)\)-partitionable by Theorem 16. If, on the other hand, the procedure never produces a vertex with weight larger than \( U \), then Theorem 16 guarantees that \( Q \) is \((L, U)\)-partitionable. If one wants to find an actual \((L, U)\)-partition of \( Q \), all what one has to do is to execute an arbitrary scan-and-cut procedure, say the greedy one, on the preprocessed path \( Q^* \). In view of Lemma 15, the procedure will output an \((L, U)\)-partition of \( Q^* \). At this point, notice that every vertex of \( Q^* \) corresponds to a subpath of \( Q \). If during the preprocessing procedure one labels the vertex obtained by compressing vertices \( i \) and \( j \) by \( \min\{i, j\} \), it is easy to “undo” the compressions so as to get \( Q \) back. Then the desired \((L, U)\)-partition of \( Q \) can be obtained simply by cutting the same edges that were cut in \( Q^* \). It is also worth pointing out that the characterizations of \((L, U)\)-partitionable paths provided by Theorem 16 rely only on a left-theory (or, symmetrically, on a right-theory). By allowing for contractions of both left-illegal and right-illegal edges, one can obtain a further characterization of \((L, U)\)-partitionable paths (see Theorem 17 below) and, even more important, a characterization of forbidden edges (see Theorem 18 below).

Let \( \sigma \) be the operator that maps every path \( Q \) into the unique stable compression \( Q^* \) of \( Q \), and let \( \delta \) be the operator that reverses the order of the vertices of \( Q \).

Consider the path \( Q^* = \delta \sigma \delta \sigma Q \): in other words, one can obtain \( Q^* \) by executing the right-to-left preprocessing procedure on \( Q \), thereby obtaining \( Q^* \); and then by executing on \( Q^* \) a left-to-right preprocessing procedure completely symmetric with respect to the above one (only the roles of “left” and “right” are interchanged).

**Theorem 17.** A standard path \( Q \) is not \((L, U)\)-partitionable if and only if \( Q^* \) has only one vertex, and this vertex has weight larger than \( U \).

**Proof.** The “if” part follows from Lemma 13 (and from its right-counterpart). Conversely, assume that \( Q \) is not \((L, U)\)-partitionable. Then by Theorem 16(4) the first node of \( Q^* \) has a weight larger than \( U \). It follows that the first edge of \( Q^* \) is right-illegal. The left-to-right procedure will then compress the first two nodes of
$Q^*$ into a single node having again a weight larger than $U$. In the new path, the first edge is again right-illegal, and so on. Therefore the left-to-right procedure will shrink $Q^*$ into a single node whose weight is obviously larger than $U$. □

**Theorem 18.** For a standard path $Q$, the nonforbidden edges are precisely the edges of $Q^*$.

**Proof.** Clearly, any edge of $Q$ which does not survive in $Q^*$ was compressed during the preprocessing phase, and hence is forbidden. Conversely, given any edge $e = (i, i+1)$ of $Q^* = 1 \ldots q$, we must prove the existence of an $(L, U)$-partition in which the edge $e$ is cut. To obtain one such partition, it suffices to execute the left greedy procedure on the path $(i + 1)(i + 2) \ldots q$; and the right greedy procedure on the path $i(i-1) \ldots 1$. By Lemma 15 (and its right-counterpart) neither procedure can get stuck. □

So far we have imposed no restrictions on the number of components of an $(L, U)$-partition. Now we want to find $(L, U)$-partitions:

1. with a minimum number $r$ of components;
2. with a maximum number $s$ of components;
3. with a given number $p$ of components, $r \leq p \leq s$.

As in Section 1, let us denote the "$(L, U)$-partition into $p$ components" problem as $\Pi_p(L, U)$. In principle, one might wonder whether a $p$-partition exists for any given $p$ such that $r \leq p \leq s$. As we shall see, the answer is always affirmative. Again, we need to use only left-concepts. We assume that the standard path $Q$ is $(L, U)$-partitionable, and that $Q$ (possibly after preprocessing) is stable. We start with the following observation.

**Lemma 19.** Assume that a given scan-and-cut procedure assigns two consecutive cuts to the edges $(i-1, i)$ and $(j-1, j)$, $j > i$; and that the greedy procedure assigns two consecutive cuts to the edges $(m-1, m)$ and $(q-1, q)$, $q > m$. Then if $i \leq m$ one must have $j \leq q$.

**Proof.** The statement follows at once from the definition of greedy procedure. □

**Theorem 20.** The greedy procedure outputs an $(L, U)$-partition with the smallest number of components.

**Proof.** For any given $(L, U)$-partition $\pi$ there is a scan-and-cut procedure which outputs $\pi$. Consider the partition $\pi'$ obtained via the greedy procedure. If $(i, i+1)$ is the first edge being cut in $\pi$ and $(i', i'+1)$ is the first edge being cut in $\pi'$, one must have $i \leq i'$ by the definition of greedy procedure. Thus, after the first cut, the greedy procedure has the lead. By repeatedly applying Lemma 19, one sees that the greedy procedure keeps the lead after the second, the third, ... cut. Hence the greedy procedure takes the checkered flag. Therefore $\pi$ has at least as many cuts as $\pi'$. □
Theorem 21. The stingy procedure outputs an \((L, U)\)-partition with the largest number of components.

Proof. Similar to Theorem 20. □

Remark. If edge \((l, l+1)\) is cut in the greedy partition and \(m\) is the largest index for which \(\sum_{k=l+1}^{m} w_k \leq U\), then the condition \(\sum_{k=l+1}^{m} w_k \geq L\) must also necessarily hold, else edge \((l, l+1)\) would be illegal, contrary to our assumption that the path \(Q\) is stable. Hence during the execution of the greedy procedure one can ignore the lower bound on the accumulated weight. Similarly, during the execution of the stingy procedure, it is not necessary to take into account the upper bound on the accumulated weight.

Recall that we have denoted by \(r\) and \(s\) the smallest and the largest number of components in an \((L, U)\)-partition, respectively. An \(r\)-partition \([\text{an } s\text{-partition}]\) will be called a min-partition \([\text{a max-partition}]\) and its cuts will be called min-cuts \([\text{max-cuts}]\).

We turn now our attention to the existence of an \((L, U)\)-partition with a prescribed number \(p\) of components. We assume that \(s \geq r + 2\) and that \(r < p < s\), in order to avoid easy cases.

Theorem 22. For an arbitrary \(p\) such that \(r < p < s\), there exists always an \((L, U)\)-partition of the path \(Q\) into \(p\) components. Furthermore, there exists an \((L, U)\)-partition \(\pi\) of \(Q\) into \(p\) classes, which has the following properties:

1. \(\pi\) is an "hybrid" of a max-partition and of a min-partition, in the sense that each cut of \(\pi\) is either a max-cut or a min-cut (or both);
2. there is a vertex \(m\) such that all cuts of \(\pi\) on the left of \(m\) are max-cuts, while all cuts of \(\pi\) on the right of \(m\) are min-cuts.

Note: some of the cuts on the left of \(m\) might be also min-cuts; similarly, some of the cuts on the right of \(m\) might be max-cuts as well.

In the following of this section we will prove Theorem 22 and we will analyze the complexity of problems \(\Pi(L, U)\), \(\Pi_{\text{min}}(L, U)\), \(\Pi_{\text{max}}(L, U)\) and \(\Pi_p(L, U)\). We keep assuming that the path \(Q\) is stable. Define, for every vertex \(i\) of \(Q\) \((i = 1, \ldots, n)\): \(a_i = \) number of max-cuts on the left of \(i\); \(\beta_i = \) number of min-cuts on the left of \(i\); \(\delta_i = a_i - \beta_i\). Clearly one has:

\[
\delta_1 = 0, \quad \delta_n = s - r, \quad -1 \leq \delta_{i+1} - \delta_i \leq 1 \quad (i = 1, \ldots, n-1).
\]

Let us introduce the set:

\[
S = \{i: \delta_i = p - r \text{ and edge } (i-1, i) \text{ bears a max-cut}\}.
\]

Lemma 23. \(S\) is nonempty.
Proof. Taking into account the constraints on the values of $\delta_i$, and using a discrete "Bolzano–Weierstrass theorem", one sees that the set $T = \{i: \delta_i = p - r\}$ is non-empty. Let $i^* \geq 2$ be the smallest element of $T$. The edge $(i^* - 1, i^*)$ must bear a cut, else one would have $\delta_{i^* - 1} = p - r$, against the minimality of $i^*$. The cut on that edge cannot be a min-cut, for otherwise one would have $\delta_{i^*} = p - r + 1$, and in this case there would be a $i^{**} < i^*$ such that $\delta_{i^{**}} = p - r$, again contradicting the minimality of $i^*$. Hence $i^* \in S$. \(\qed\)

Let $m = \max\{i: i \in S\}$; clearly one has $m < n$.

Lemma 24. The first cut on the right of $m$ is a max-cut, and it cannot be a min-cut at the same time.

Proof. Call $(q - 1, q)$, with $q > m$, the first edge on the right of $m$ bearing a cut. Such cut cannot be a min-cut, else one would have $\delta_q = p - r - 1$ and then there would be an index $j^* > q$ such that $\delta_{j^*} = p - r$, against the maximality of $m$. Hence the cut on the edge $(q - 1, q)$ must be a max-cut. \(\qed\)

Remark. By the definition of $m$, edge $(m - 1, m)$ bears a max-cut. It is quite possible that such edge bears also a min-cut.

We can now conclude the proof of Theorem 22.

Proof of Theorem 22. Let $\pi$ be the partition whose cuts are all the max-cuts on the left of vertex $m$, followed by all min-cuts on the right of $m$.

Claim 1. $\pi$ is a $p$-partition.

By definition, $\delta_m = p - r$. Hence the total number of cuts of $\pi$ is $\alpha_m + (r - 1 - \beta_m) = \delta_m + r - 1 = p - r + r - 1 = p - 1$. Therefore $\pi$ is a $p$-partition.

Claim 2. $\pi$ is an $(L, U)$-partition.

Every class of $\pi$ coincides either with a class of the max-partition or with a class of the min-partition, with the possible exception of the hybrid class $C$ whose left border is the max-cut on edge $(m, m - 1)$, and whose right border is the first min-cut on the right of $m$ (if there is no min-cut on the right of $m$ then $C$ has no right border). Therefore all we have to prove is that $L \leq \sum_{h \in C} w_h \leq U$. Let $i$ be the maximum index $\leq m$ such that edge $(i - 1, i)$ bears a min-cut (if there are no min-cuts on the left of $m$ we set $i = 0$). Let $j$ be the minimum index $> i$ such that edge $(j - 1, j)$ bears a min-cut (if no such edge exists then we set $j = n + 1$). Finally, let $q$ be the minimum index $> m$ such that edge $(q - 1, q)$ bears a max-cut (such edge always exists). By Lemma 24 one must have $j > q$. By definition, $C = \{m, m + 1, \ldots, j - 1\}$. Moreover, $L \leq \sum_{h = m, \ldots, q - 1} w_h \leq \sum_{h = m, \ldots, j - 1} w_h = \sum_{h \in C} w_h$ and $U > \sum_{h = i, \ldots, j - 1} w_h \geq \sum_{h = m, \ldots, j - 1} w_h = \sum_{h \in C} w_h$. This completes the proof. \(\qed\)
Complexity analysis. As mentioned in the previous sections (see in particular Section 2), one can check whether the weighted path \((Q, w)\) is \((L, U)\)-partitionable by a right-to-left preprocessing procedure. Let us examine the time complexity of such procedure. During its execution, every vertex of \(Q\) enters and leaves the queue QUEUE. The length of QUEUE is variable, and one can actually build examples where such length gets very close to its trivial upper bound \(n\). However, during the execution of the procedure, a vertex which is deleted from QUEUE never enters QUEUE again, i.e., every vertex enters and leaves QUEUE exactly once. It follows that right-to-left preprocessing requires \(O(n)\) time.

It is also clear that, after the path has been preprocessed, both the greedy and the stingy procedures require \(O(n)\) time. Once a minimum and a maximum partition are at hand, it is easy to obtain in \(O(s)\) time an \((L, U)\)-partition into any intermediate number of components. According to Theorem 22, all what one has to do is to compute the index \(m\) (see Section 2)—this can be obviously done in \(O(s)\) time—and then to generate, again in \(O(s)\) time, all max-cuts on the left of vertex \(m\) and all the min-cuts on the right of \(m\). In conclusion, one can find an \((L, U)\)-partition into any given number \(p\) of components also in \(O(n)\) time.

5. Most uniform partitioning

The theoretical framework developed in the previous sections can be used to solve the most uniform partitioning problem (MUP) as introduced in the first section. A set of problems strictly related to MUP will also be considered.

The solution of all the problems introduced in the following will be based on the solution of the "\((L, U)\)-partitioning into \(p\) components" problem \(\Pi_p(L, U)\) analyzed.
in the previous sections. In particular it is important to notice that this problem can be solved in time \( O(n) \) whether \( p \) is fixed or not.

Let us denote by \( F(p) \) the feasible region of \( \Pi_p(L, U) \) for \( L \) and \( U \) in the interval \([0, W]\), i.e., \( F(P) \) is the set of pairs \((L, U)\) such that the answer to \( \Pi_p(L, U) \) is "yes", and by \( NF(p) \) the infeasible region of \( \Pi_p(L, U) \), i.e., the set of pairs \((L, U)\) such that the answer to \( \Pi_p(L, U) \) is "no". The structure of the sets \( F(p) \) and \( NF(p) \) is given in Fig. 8 (where \( NF(p) \) corresponds to the white area in the square \(([0, 0], (W, 0], (0, W], (W, W)]\)); remark that \( F(p) \) is always nonempty, since \((0, W)\) is feasible for any \( p \leq n \).

The structure follows immediately from the following easy-to-prove facts, which hold for arbitrary graphs:

1. If a pair \((L', U')\) is feasible, then any pair \((L, U)\) with \( L \leq L' \) and \( U \geq U' \) is also feasible; in this case we shall also say that \((L', U')\) dominates \((L, U)\); if there exists no pair \((L, U)\) which dominates \((L', U')\), then we say that \((L', U')\) is a dominant pair; remark that, given any two dominant pairs \((L', U')\) and \((L'', U'')\) with \( L' < L'' \), one has \( U' < U'' \).

2. For every fixed \( U \), the maximum \( L \) such that the pair \((L, U)\) is feasible coincides with the weight of a component; similarly if one minimizes \( U \) with fixed \( L \); therefore the coordinates of a dominant solution can take only a finite number of values, in fact no more than \( 2^n \) values.

Facts (1) and (2) lead to the staircase form of the contour separating \( F(p) \) from \( NF(p) \); the dominant solutions are precisely the corners of the staircase; in Fig. 8 the points \((D1, D2, D3)\) are the corners.

3. If the graph is partitioned into \( p \) components, the average weight of each component is equal to \( W/p \); therefore any pair \((L, U)\) such that either \( L > W/p \) or \( U < W/p \) (or both) is infeasible, i.e., for any feasible pair \((L, U)\) one must have \( L \leq W/p \leq U \); this implies that the feasible region must belong to the subrectangle \([0, W/p], (W/p, W), (0, W), (W/p, W)]\).

4. All feasible pairs belonging to a same line \( U = L + c \) have the same value of the objective function \( z = c \); therefore the most uniform solution corresponds to a solution on the line with minimum \( c \), i.e., the line nearest to the diagonal \([0, 0], (W, W)\) of the square; in Fig. 8 the point \( D1 \); remark that the most uniform solution is always a corner.

5. If \( p \) is not fixed the solution of the MUP is trivially given by the unique component \((p = 1)\) corresponding to the point \((W, W)\) in the \((L, U)\)-diagram, with objective function \( z = 0 \).

6. The following relation among \( U, L, W \) and \( p \) holds: \( U - L \leq \min\{pU - W, W - pL\} \); it follows that if either \( L = W/p \) or \( U = W/p \), then \( L = U \), i.e., in the first row and the last column of the subrectangle containing the feasible region, the point \((W/p, W/p)\) is the only candidate to be a corner (in this case it is the unique corner).

In the case of paths it is possible to add one more fact:

7. Only \( O(n^4) \) points corresponding to feasible \((L, U)\)-pairs need to be considered; in fact there exist only \( O(n^2) \) possible component weights (more precisely
most uniform path partitioning

253

\( n(n+1)/2 \), therefore only \( O(n^2) \) values for \( L \) and \( U \) may be considered; as fact (1) still holds, it follows that there exist at most \( O(n^2) \) corners which are candidates to be optimal.

On this ground we can now propose a simple strategy to solve two basic problems obtained as variants of MUP: MAXMIN and MINMAX.

**MAXMIN.** Given \( U \), find \( \pi = \{C_1, C_2, \ldots, C_p\} \) such that:

(i) \( L \leq W(C_i) \leq U \), for \( i = 1, 2, \ldots, p \);

(ii) \( L \) is maximum.

**MINMAX.** Given \( L \), find \( \pi = \{C_1, C_2, \ldots, C_p\} \) such that:

(i) \( L \leq W(C_i) \leq U \), for \( i = 1, 2, \ldots, p \);

(ii) \( U \) is minimum.

A solution procedure for the two above problems is based on binary search performed on horizontal or vertical lines of the \((L, U)\)-plane. The number of points which are candidates to be optimal in each row or column of the \((L, U)\)-plane is \( O(n^2) \). Therefore such a procedure finds the solution in time \( O(n \log n) \) by solving \( O(\log n) \) problems \( \Pi_q(L, U) \). This approach can be used to solve MUP. In fact it is possible to solve alternately MAXMIN and MINMAX following the borderline dividing regions \( \mathcal{F}(p) \) and \( \mathcal{N}(p) \). If \( q \) is the number of corners, then the complexity of such a procedure is \( O(nq \log n) \), since the solution of \( q \) MAXMIN or MINMAX problems is required. \( q \) cannot obviously be larger than the number of pairs \((L, U)\) lying on the borderline, i.e., \( O(n^2) \). Actually, it turns out that \( q \) can never exceed \( pn \).

**Theorem 25.** \( q \leq pn \).

**Proof.** Let \( F_k(j) \) be the "yes" region obtained by partitioning into \( k \) components the subpath \( \{1, \ldots, j\} \) and let \( f_k(j) \) be the number of corners of \( F_k(j) \); let \( V_k \) be the set of corners of \( \mathcal{V}F_k = F_k(k) \cup F_k(k+1) \cup \cdots \cup F_k(n) \) (we assume in the following that \( p \) can be any number between 1 and \( n \)) and \( v_k = |V_k| \). Let \( F_k(j \mid i) \) (with \( i < j \)) be the "yes" region obtained by partitioning into \( k-1 \) components the subpath \( \{1, \ldots, i\} \) and adding the subpath \((i+1, \ldots, j)\).

We will prove that:

(i) \( v_1 = n, v_2 = n \) (moreover: \( v_n = 1, f_1(j) = 1, f_2(j) = 1, j = 1, 2, \ldots, n \));

(ii) \( v_k \leq v_{k-2} + n, k = 3, \ldots, n \);

(iii) \( f_k(j) \leq v_k, k = 3, \ldots, n, j = k, \ldots, n \).

The thesis, i.e., \( q = f_p(n) \leq np \), follows directly from those relations.

(i) and (iii) are trivial. In fact each \( F_1(j) \) produces exactly one corner located on the diagonal \(((0,0),(W, W))\) with values of \( L \) and \( U \) given by the sum of the weights of the first \( j \) vertices of the path; \( F_p(n) \) produces one corner with \( L = \min_j w_j \) and \( U = \max_j w_j \). Moreover, let \( w(i, j) \) be the sum of the weights of vertices \( \{i, \ldots, j\} \).
of the path and let $R(i+1, j)$ be the rectangle with vertices $\{(0, w(i+1, j)), (0, W), (w(i+1, j), 0), (w(i+1, j), W)\}$; it is easy to verify that the corners of $F_2(j \mid i)$, $i = 1, \ldots, j-1$, are all on the diagonal of $R(i+1, j)$; therefore $F_2(j)$, $j = 1, \ldots, n$, has only one corner and $u_2 = n$.

To prove (ii) remark that:
- the “yes” region corresponding to the 2-partition of any subpath $(i+1, \ldots, j)$ is a rectangle $R2(i+1, j)$;
- $V_F = \bigcup_{j=k, \ldots, n} F_k(j) = \bigcup_{j=k, \ldots, n} \left( \bigcup_{i=k-2, \ldots, j-2} [F_{k-2}(i) \cap R2(i+1, j)] \right)$.

In fact, the corners of each intersection are given by a subset of the corners of $F_{k-2}(i)$ plus at most two new corners due to the intersection. Remark that these new corners replace at least one corner of $F_{k-2}(i)$ eliminated by the intersections. Therefore all the corners $V_k$ are given by a subset of the corners of $V_{k-2}$ plus a new set of corners due to the intersections; but these new corners are at most one for each $i$, and hence at most $n$. In conclusion, $u_k \leq u_{k-2} + n$.

The overall worst-case performance of the previous procedure is $O(n^2p \log n)$.

In “Procedure MUP”, at the end of this section, a Pascal-like version of the solution procedure is given.

A procedure can be given to solve MUP without binary search. In fact it is possible to follow the borderline and to solve all problems lying on it. The number of such
problems cannot exceed the number of pairs \((L, U)\) lying on the borderline, i.e., \(O(n^2)\). Therefore the complexity of such a procedure is \(O(n^3)\). As a matter of fact this procedure is basically the same as the one described in [11]. With reference to Fig. 9 the procedure amounts to solve the sequence of problems \(\Pi_p(L, U)\): \(\{1, 2, \ldots, 24\}\); point \(\{13\}\) corresponds to the most uniform partition.

A more efficient procedure (with the same worst-case figure) can be given. In fact, to find the most uniform partition, it is possible to follow, instead of the borderline, a line obtained by following, from each corner, a 45° line leaving on its left all solutions corresponding to worse partitions. With reference to Fig. 9, the sequence of problems solved is \(\{1, 2, 3, 4, 5, 6, 8, 9, 10, 11, 12, 13, 14, 15, 18, 25\}\).

Some particular cases of MUP are easy to solve.

- **\(p = 2\)**. If \(p = 2\) then there exists a unique corner. An optimal solution to MUP can therefore be found in time \(O(n \log n)\) by solving \(O(\log n)\) problems \(\Pi_p(L, U)\). Furthermore the solution obtained is, in this case, optimal for the following problems: optimal equi-partitioning into two components with \(L_1\)-norm, minmax and maxmin partitioning into two components.

- **\(L = U\)**. For any graph, if \(L = U\), then there exists a feasible solution only if \(L = W/p\) and, if the graph is a tree, the feasible solution, if any, can be found in time \(O(n)\) by a greedy algorithm which cuts off a component of weight \(W/p\) at each step.

**Procedure MUP.**

*Input:* the weights \(\{w_1, \ldots, w_n\}\).

*Output:* the corners \(\{(L_1, U_1), \ldots, (L_q, L_q)\}\).

\(L_0 := 0; \quad i := 1;\)

while \((U = W)\) do

begin

Vertical search: Binary search on the column \(L = L_{i-1}\) to find the “yes” pair with minimum \(U\), let \(U_i\) be the output;

Horizontal search: Binary search on the row \(U = U_i\) to find the “yes” pair with minimum \(L\), let \(L_i\) be the output;

\(i := i + 1;\)

end

**Acknowledgement**

We thank Professors Rainer Schrader and Jarik Nešetřil for their useful comments.

The financial support from the “Ministero della Pubblica Istruzione”, Italy, is gratefully acknowledged.

Sections of this paper were written while the second author was visiting the De-
partment of Statistics of the University “La Sapienza”, Rome; and while the third author was visiting the Forschungsinstitut für Diskrete Mathematik of the Bonn University.

References