Journal of Classification 21:207-230 (2004)
DOI: 10.1007/s00357-004-0017-9

Improving Dynamic Programming Strategies for Partitioning

B.J. van Os and J.J. Meulman

Leiden University, The Netherlands

Abstract: Improvements to the dynamic programming (DP) strategy for partitioning (nonhierarchical classification) as discussed in Hubert, Arabie, and Meulman (2001) are proposed. First, it is shown how the number of evaluations in the DP process can be decreased without affecting generality. Both a completely nonredundant and a quasi-nonredundant method are proposed. Second, an efficient implementation of both approaches is discussed. This implementation is shown to have a dramatic increase in speed over the original program. The flexibility of the approach is illustrated by analyzing three data sets.

Keywords: Cluster analysis; Partitioning; Dynamic programming; Exact algorithm.

1. Introduction

The application of dynamic programming to a data analysis problem in classification can be found as early as Fisher (1958). Fisher describes an algorithm for the partitioning problem, where the objects within each class must be contiguous with respect to a given object order. Because this algorithm solves the problem in polynomial time, the algorithm has become quite popular and has been reinvented several times (Olstad and Manne 1995; Lim and Lee 1997; Alpert and Kahng 1997). Moreover, a few efficiency improvements have been suggested (Olstad and Manne 1995), and the method itself has been used in an heuristic approach to solve large partitioning problems by imposing an empirically obtained order on the objects (Alpert and Kahng 1995, 1997).

Supported by The Netherlands Organization for Scientific Research (NWO) by grant No. 030-56-403 for the 'PIONEER' project 'Subject Oriented Multivariate Analysis' to the second author.

Authors' Address: Department of Educational Studies, Data Theory Group, Leiden University, P.O. Box 9555, 2300 RB Leiden, The Netherlands, e-mail: os@fsw.leidenuniv.nl

The formulation of a DP recursion for partitioning (hereafter DPP) without any contiguity constraints on the objects was first suggested by Jensen (1969). Not surprisingly, given current desktop computing power, DP methods have recently met a renewed interest, e.g., see the recent comprehensive overview of old and new applications of DP to combinatorial data analysis given by Hubert, Arabie, and Meulman (2001). Using DP strategies, these authors also offer a series of programs including several that deal with restricted and unrestricted partitioning problems.

An overview of other mathematical programming techniques for partitioning problems can be found in Hansen and Jaumard (1997). A technique closely related to dynamic programming is branch-and-bound, which is another partial enumeration technique. Where DP reduces the solution space (as compared to complete enumeration) by exploiting the principle of optimality, branch-and-bound tries to reduce the solution space by exploiting bounding equations, specific to the objective function (a particular cluster criterion). Branch-and-bound has been applied to partitioning problems such as within sum-of-squares partitioning (also known as K-means) (Koontz, Narendra, and Fukunaga 1975; Diehr 1985), within sum of dissimilarities partitioning (Klein and Aronson 1991; Brusco 2003), and minimum-diameter partitioning (Hansen and Delattre 1978). Success of the branch-and-bound approach depends upon the quality of the bounds and the particular data set that is analyzed, and running time is not known in advance. However, branch-and-bound does not need a large amount of storage space such as DP and therefore can handle larger problem instances. In contrast, the appeal of the DP approach to partitioning certainly lies in its flexibility to handle a broad range of homogeneity criteria optimally — with and without restrictions — and its current ability to cope with nontrivial problem sizes in a priori known running time. Furthermore, Hubert et al. (2001) have successfully shown the possibility of extending the original DP approach heuristically to handle even larger problem sizes. In this paper a new, more efficient DP formulation for partitioning will be presented.

2. The Dynamic Programming Recursion for Partitioning

Suppose we have the set partitioning problem: given a set of N objects, represented by their indices $S = \{1, \ldots, N\}$, find a collection of K mutually exclusive and exhaustive subsets of S, say $\{S_1, \ldots, S_K\}$, defining the *partition* $\mathcal{P}$, such that some objective function $f(\mathcal{P})$ is optimized over $\mathcal{P} \in \Phi$, the set of feasible partitions. Furthermore, it is assumed that we have defined some homogeneity (or heterogeneity) function on the subsets and the data; for an arbitrary subset S_k, this function will be denoted by $h(S_k)$. For dynamic programming to work, the function $f(\cdot)$ is required to be monotone over the subset homogeneity functions $h(S_k)$. Here we will assume an additive function for $f(\cdot)$, and have the overall mathematical program

$$\begin{cases} \operatorname*{opt}_{\mathcal{P}} \sum_{k=1}^{K} h(S_k), \\ \text{subject to} \\ S_k \in \mathcal{P} \in \Phi \\ S_k \neq \emptyset \\ S_k \cap S_{k'} = \emptyset \text{ for } k \neq k', \\ \bigcup_{k=1}^{K} S_k = S. \end{cases} \tag{1}$$

Suppose we let $f_k^*(S')$ denote the optimal value for partitioning the set $S' \subseteq S$ into k clusters. Then, dynamic programming provides an exact algorithm by exploiting the recursion

$$f_k^*(S') = \begin{cases} f_k^*(S') = \operatorname*{opt}_{S_k \subset S'} (f_{k-1}^*(S' - S_k) + h(S_k)) \text{ for } k > 1, \\ f_1^*(S') = h(S'), \end{cases} \tag{2}$$

subject to the constraints of the mathematical program. In the simplest case when Φ is the full set of partitions, these constraints imply that in (2) any cluster $S_k \subset S'$ does not exceed the size bounds

$$1 \leq |S_k| \leq |S'| - (k-1). \tag{3}$$

The overall optimal value for the mathematical program is given by $f_K^*(S)$.

This recursion is used both by Jensen (1969) and Hubert, Arabie, and Meulman (2001) (HAM for short). Although the recursion immediately leads to an implementable algorithm, efficient DP algorithms start at the first level, where $k = 1$, and the formulation of such a backward algorithm is nontrivial. Both the approach of Jensen and the one used by HAM leave part of the potential unexplored. The issue is that although the recursion (2) is correct, it is much too general. Any particular subset, say $\{1, 4, 7\}$, will be enumerated for all clusters $\{S_k | k = 1, \ldots, K\}$, and evaluated in all stages of the recursion. The order of the particular clusters in the final partition is irrelevant, however, and such generality is not needed. An immediate implementation of this recursion consequently involves redundancies in the entities processed in the enumerative processes. An efficient algorithm requires a nonredundant definition of the enumeration processes involved. This definition will be the main focus of this paper. Moreover, for a successful implementation of such an algorithm, both the enumerative processes and the storage and retrieval of optimal values must be implemented very efficiently. An example of such an implementation will be given.

2.1 Jensen's Formulation

Jensen (1969) was the first to introduce a dynamic programming formulation for an unrestricted partitioning problem, where the objective function was defined as in (1) for a particular homogeneity function expressing the homogeneity of a cluster by the average squared distances between the objects in the cluster. Jensen goes through significant detail in defining the number of feasible *states* (the number of instances of $S_k, S_k \subset S'$ in the recursion) and the total number of feasible *arcs* in the DP network (the number of feasible transitions from a feasible state in stage k to a feasible state in stage $k+1$), using the general recursion. He notes that in this formulation quite some redundancy occurs due to symmetry, and defines the maximum number of feasible arcs required to optimally solve the problem in his formulation, denoted as NAT. In other formulations hereafter, the number of arcs will be phrased as the number of *recursive evaluations*.

Jensen shows how in principle some redundancies can be removed. A distribution form of a partition is a sequence of numbers denoting the cluster sizes of an ordered sequence of clusters. For example, for clustering seven objects into 3 clusters, examples of distribution forms are: $(5)(1)(1)$, $(4)(2)(1)$, $(3)(3)(1)$, etc. As shown above, Jensen proposes to only consider partitions according to distribution forms where the cluster sizes decrease from left to right. Furthermore, he notes that in the recursive process at stage $k = 2$, half of the transitions can be left out: if we evaluate the recursion for some $S_k \subset S'$, we can ignore the evaluation of $\overline{S_k}$, the complement set. Jensen even suggests elimination of some more redundancies for stage $k = 3$ explicitly by looking at distributional forms; however, for growing K and N, removing redundancies by such a process becomes cumbersome, and implementation requires a lot of bookkeeping. It must be noted that Jensen does not give a complete algorithm description, nor a program to execute the DP process. The two subprograms offered can not simply be combined to construct an efficient implementation of his ideas. Moreover, as we will see in the final comparison of the effectiveness of several approaches, his formulation still leaves a lot of redundancies in the process.

Dodge and Gafner (1994) have further elaborated on the idea of distribution forms. Their approach incorporates a relaxation of the original problem leading to a reduction in the number of evaluations needed, circumventing the occurrence of redundancies. Their approach does not guarantee the optimal solution of the original problem, as does the general mathematical program considered in this paper.

2.2 The HAM Approach to Partitioning

Hubert, Arabie, and Meulman (2001) were the first to give a complete FORTRAN program (DPCL1U) for the general partition problem (1), including mini-max objective functions. They do not consider detailed procedures to avoid redundancies in the recursive process, except for two aspects of the process. First, they avoid recalculating subset homogeneities/heterogeneities by identifying two phases in the DP process. In the first phase (Stage 1), a homogeneity/heterogeneity measure $h(S_k)$ is evaluated for all $2^N - 1$ possible subsets S_k. The measures used by HAM require on average for each subset a number of operations of N^2 (average dissimilarity) up to approximately N^3 (comparing dissimilarities within versus between clusters); therefore the order of this phase is either $O(N^2 2^N)$ or $O(N^3 2^N)$[1]. All these subset homogeneities are stored and retrieved in later stages. Second, they propose proceeding through the recursion to obtain all optimal partitions of the full set S into k clusters, $2 \leq k \leq K$.

The rationale behind their approach is that although at Stage K we are only interested in an optimal partitioning of the full set (i.e. $S' = S$), for evaluating this $f_K^*(S)$ we will have to refer to all optimal partition values $f_{K-1}^*(B)$ into $K - 1$ clusters of all subsets $B \subseteq S$. Therefore, we have to enumerate all $S' \subseteq S$, while enumerating for each S' all possible partitions into two subsets A and B, and evaluate the recursion (2) for all those bipartitions. The number of recursive evaluations made in the algorithm is

$$\sum_{s=k-1}^{N} \binom{N}{s} \left(2^{N-s} - 1\right). \tag{4}$$

In the last stage where $k = K$, only $S' = S$ is considered and the number of evaluations is $\binom{N}{2}$. The basic enumerative process is therefore of the type "generate all subsets out of all subsets" and the order of such a process is $O(3^N)$. Note that this order is considerably higher than for Stage 1 (calculating all subset homogeneities). Because at each stage the subset $S' = S$ is also evaluated, all optimal partitions in less than K clusters are also given. In a sense, this can be viewed as avoiding redundant calculations, an idea maintained in the following sections. There are, however, quite a number of redundancies left in this approach, and several alternatives for removing those as well as a completely nonredundant formulation will be presented in Section 2.4. The actual

[1] A function $g(n)$ is said to be *of order* $f(n)$, denoted by $g(n) = O(f(n))$, if there exists positive constants c_0 and n_0 such that $g(n) < c_0 f(n)$ for all $n > n_0$; for an introduction, see Day (1996).

implementation used in DPCL1U will also be greatly improved.

2.3 Removing Redundant Evaluations by Applying Size Bounds

The symmetry in finding the solution through the recursion is most prominently seen when evaluations are done at Stage 2, where $f^*_{k-1}(S' - S_k)$ is in fact $h(S' - S_k)$, but it occurs at higher stages as well. A practical perspective for exploiting such symmetry is to define bounds for the sizes of subsets involved in evaluating the recursion (without looking explicitly at *distribution forms*). First, consider Stage 2 and suppose we evaluate a partitioning of S' into A and B; if A has size $|A| = s$ ($s < N$), then B has to be of size $|B| = N - s$. If we would evaluate all partitions involving A of size $1 \leq |A| \leq s$, we evaluate at the same time all partitions involving B of size $N - s \leq |B| \leq N - 1$. Therefore, all permutations of 2 clusters out of S' (i.e. respecting cluster order) can be defined by defining all $A \subseteq S'$ of size $1 \leq |A| \leq N - 1$. By exploiting the above mentioned symmetry, all partitions into 2 clusters of S' (without respecting cluster order) can be defined by all subsets $A \subseteq S'$ of size $1 \leq |A| \leq \left[\frac{N}{2}\right]$ (we adopt here Jensen's notation $\left[\frac{i}{j}\right]$ to denote the integer part of $\frac{i}{j}$).

We can extend this approach to stages $k > 2$ by noticing that if an upper bound of $\left[\frac{N}{k}\right]$ is provided for $|A|$, all partitions involving A for $\left[\frac{N}{k}\right] + 1 \leq |A| \leq \left[\frac{N}{2}\right]$ are implicitly evaluated since they are assessed as a part of finding the *optimal* partitioning S' into $k - 1$ partitions at a previous stage. We can define an even stricter upper bound for $|A|$ if we devise the algorithm such that it first generates subsets S', and then subsets A. In that case, the upper bound for $|A|$ can be defined as $\left[\frac{|S'|}{k}\right]$.

After exploiting the symmetry by using the size bounds on A, the amount of work can be further decreased by working backwards. At Stage K, we do not have to evaluate partitions of all $S' \subseteq S$, because we are interested in an optimal partitioning of S. Therefore, we can define a lower bound for $|S'|$ at stages $k < K$ by

$$L(k) = L(k+1) - \left[L(k+1)/k + 1\right], \tag{5}$$

where $L(K) = N$. The upper bound $U(k)$ is given by $U(k) = N - (K - k)$ if we are only interested in an optimal partition into K clusters, and by $U(k) = N$ for all k if we want all optimal partitions into k clusters, $1 < k < K$. This results in a total number of evaluations at Stage k of respectively either

$$\sum_{s=L(k)}^{N-(K-k)} \left\{ \binom{N}{s} \sum_{s'=1}^{\left[\frac{s}{k}\right]} \binom{s}{s'} \right\}, \tag{6}$$

or

$$\sum_{s=L(k)}^{N} \left\{ \binom{N}{s} \sum_{s'=1}^{\left[\frac{s}{k}\right]} \binom{s}{s'} \right\}. \tag{7}$$

Section 2.5 gives some totals for the different processes for selected problem sizes. Note that (6) and (7) are both smaller than Jensen's NAT, except for $K = 3$ and $N < 9$ where Jensen explicitly removes all redundancies. One should also note that (7) is much larger for $k = 2$ than for $k > 2$ if N becomes large. Therefore the workload of obtaining all optimal partitions up to K clusters is only slightly dependent on the number of clusters K. Also, the total number of evaluations given by (6) for large N *decreases* if K becomes large.

2.4 A Formulation Completely Nonredundant with Respect to Cluster Order

Although we so far have been able to avoid many redundancies by applying size bounds, some redundancies are still left unexploited. We will now develop a general way of avoiding redundancies due to the irrelevance of the order of clusters in a partition for all stages. First, consider Stage 2 and define two complementary sets, Ω_A being a set of subsets $A \subseteq S'$, and Ω_B being a set of subsets $B \subseteq S'$, subject to

$$|\Omega_A| = 2^{|S'|-1} - 1, |\Omega_B| = 2^{|S'|-1} - 1,$$
$$\Omega_A \cap \Omega_B = \emptyset,$$
$$(S' - A) \in \Omega_B, \forall A \in \Omega_A.$$

Constructing two such sets Ω_A and Ω_B is relatively easy. Suppose we take out one object i of S' to form S''. Then define $\Omega_A = \{A \subseteq S''\}$ and $\Omega_B = \{B = (S'' - A) + i \mid A \subseteq S''\}$. Clearly, Ω_A and Ω_B are complementary, and because all $A \subseteq \Omega_A$ do not contain i, $\Omega_A \cap \Omega_B = \emptyset$. A nonredundant enumeration of all partitions into two subsets then reduces to enumerating $\forall A \in S', B = S' - A$.

For stages $k > 2$, a non-redundant enumeration process can be accomplished by generating all partitions starting with an arbitrary order of the objects, extending the mechanism shown above for a partition into two clusters at Stage 2. Suppose we want to generate all possible partitions for complete enumeration exclusively, without generating redundancies due to different orderings. We start to define the first cluster S_1 of a partition into K groups by

$$S_1 \subset S, \text{ subject to}$$
$$\{i\} \in S_1; 1 \leq |S_1| \leq N - (K - 1),$$

where $\{i\}$ is the object with smallest index number out of S: $i = 1$. Similarly, cluster S_2 can be defined by

$$\forall S_1 : S_2 \subset S - S_1, \text{ subject to}$$
$$\{i'\} \in S_2; 1 \leq |S_2| \leq N - (K-2),$$

where i' is the object with smallest index of $S - S_1$. Using this approach, the algorithm in Table 1 enumerates all possible partitions of S into K clusters if started with $\textsc{Enumerate}(K, S)$. (The enumeration approach taken here is implicit in a recursion for Stirling numbers of the second kind given by Hartigan (1975, p. 130), and was proposed by Klein and Aronson (1991) for enumeration in a branch-and-bound algorithm.)

Fortunately, we can apply a similar mechanism to our DP partitioning recursion. When an optimal partition into exactly K clusters is needed, we can skip $K - k$ objects at each Stage $2 \leq k \leq K$, due to the exclusion of specific *smallest objects* in the enumeration procedure. The total number of evaluations for stages $k, 2 \leq k < K$ is given by

$$\sum_{s=k}^{N-(K-k)} \left\{ \binom{N-(K-k)}{s} \sum_{s'=0}^{s-k} \binom{s-1}{s'} \right\}. \tag{8}$$

This approach to obtain a partition into K clusters *only* is completely non-redundant with respect to different orderings of subsets in the enumeration processes required for recursion (2).

To obtain *all* optimal partitions into k clusters, $2 \leq k \leq K$, it is most efficient to go through all stages in a slightly different way to obtain all those partitions in one pass, in stead of repeating the above approach $k - 1$ times. This time, we can only exclude an object for cluster S_1. The total number of evaluations for Stage k becomes

$$\sum_{s=k}^{N-1} \left\{ \binom{N-(K-k)}{s} \sum_{s'=0}^{s-2} \binom{s-1}{s'} \right\}, \tag{9}$$

with the difference from argument (8) being the upper limit $s - 2$ for s'. It appears that this method to obtain partitions all partitions, $2 \leq k \leq K$, is still very efficient in removing redundancies (see Section 2.5, but it no longer upholds the nonredundancy mechanism at *all* stages and therefore will be called *quasi-nonredundant*.

2.5 Relative Efficiency of the Different Formulations

Table 2 gives a comparison of the number of evaluations needed for several DP formulations. The first column denotes the partition problem size, ex-

Table 1: Algorithm ENUMERATE(k, S'). Enumerate all possible kth clusters $S_k \in \mathcal{P}, S_k \subset S'$

```
**if** $k = 1$ **then**
    print $S_1, \ldots, S_K$;
**else**
    $\{i\}$ = the object with smallest index number in $S'$
    **for all** $\{S_k \subset S' \mid \{i'\} \in S_k; 1 \leq |S_k| \leq N - (K - k)\}$ **do**
        execute ENUMERATE$(k - 1, S' - S_k)$
    **end for**
**end if**
```

pressed in N and K. For each problem, the first row gives the number of evaluations; the second row gives a 'speedup' factor relative to the workload in the second column defined by the number of evaluations given in Jensen's formulation. Note that this is a highly theoretical number, because Jensen neither gave a precise algorithm nor a program for performing that number of evaluations. The next two columns give workloads for the two new formulations when partitioning into exactly K clusters. The last three columns give workloads for 2 to K clusters for the HAM approach, as well as the two new formulations that respectively exploit symmetry by size bounds and are quasi-nonredundant.

When it comes to partitioning into exactly K clusters, both new formulations are better than Jensen's. The nonredundant formulation is especially superior, and the advantage increases when N and K increase. The redundancy eliminations of Jensen's formulation are only effective when partitioning into a small number of clusters. When partitioning into 2 to K clusters, the quasi-nonredundant formulation is also superior compared to the HAM formulation, and the speedup factor can be very large especially when K is large. Note that in some cases (8/20 and 9/25), the size symmetrical approach is more efficient than the quasi-nonredundant approach due to the fact that when K increases beyond, say, ($N/3$), the number of evaluations of the size symmetric approach is very small for the final stages. This is a result of the symmetry size bounds, while at the same time the backward size bounds decrease the number of evaluations for earlier stages.

Two remarks are in order. First, the DP approach is not more efficient than complete enumeration for partitioning into $K = 2$ clusters. For $K = 3$, the approach is only slightly more efficient than complete enumeration because all subset homogeneities are calculated only once. Second, it must be noted that these comparisons are highly theoretical. They do express the relative efficiency of redundancy eliminations but the actual workload of programs using the approaches is highly dependent upon the implementation of these algorithms. These implementations differ substantially for the different approaches.

Table 2: Relative efficiency of DP processes for partitioning.

K/N	Jensen	Size Sym.	NR	HAM	Size Sym.	Quasi-NR
	NAT	K only	K only	all	all	all
4/10	27010	22750	11028	149311	25986	17078
	1.00	1.19	2.45	1.00	5.75	8.74
7/10	11340	10960	1840	190570	42531	13648
	1.00	1.03	6.16	1.00	4.48	13.96
4/15	9.37E6	6.88E6	3.08E6	4.14E7	7.03E6	4.66E6
	1.00	1.36	3.04	1.00	5.89	8.88
6/15	1.36E7	7.71E6	2.50E6	6.13E7	9.36E6	7.88E6
	1.00	1.77	5.44	1.00	6.55	7.78
9/15	7.87E6	3.51E6	466559	7.09E7	1.09E7	6.23E6
	1.00	2.24	16.88	1.00	6.50	11.38
5/20	3.97E9	2.09E9	8.15E8	1.37E10	2.12E9	1.71E9
	1.00	1.90	4.87	1.00	6.46	8.01
8/20	4.62E9	1.76E9	5.37E8	2.09E10	2.33E9	2.81E9
	1.00	2.63	8.61	1.00	8.97	7.44
15/20	6.74E7	4.14E7	1.30E6	2.32E10	2.45E9	5.04E8
	1.00	1.63	52.00	1.00	9.47	46.03
5/25	9.82E11	4.93E11	2.03E11	3.37E12	4.94E11	4.22E11
	1.00	1.99	4.84	1.00	6.82	7.99
9/25	1.66E12	4.37E11	1.51E11	6.13E12	5.38E11	8.65E11
	1.00	3.80	11.01	1.00	11.39	7.09
18/25	1.87E10	4.53E9	1.45E8	7.06E12	5.45E11	1.64E11
	1.00	4.12	129.0	1.00	12.95	43.05

Note: Each second row displays the speedup factors relative to the first column of each set of three columns. NR = NonRedundant process.

2.6 Efficient Implementations of the Algorithm

Implementing the DP algorithm primarily consists of carrying out a representation–enumeration problem using what is commonly referred to as Gray-Coding (Nijenhuis and Wilf 1975). This solution is based on the principle of representing sets by *atomic* vectors or *binary* representations that in turn can be seen as integers and used as ordinal numbers or indices for accessing array entries. Finding an efficient implementation turns out to be a non-trivial problem dependent on the problem size parameters. Although so far in this paper an effort has been made to theoretically remove all redundancies, an actual implementation of the non-redundant approach need not be faster than an implementation involving (some) redundancies. In practice a balance is sought between a very fast implementation with some redundant operations and a much slower implementation that is completely non-redundant. Fortunately, for problems where one partitions in a relatively small number of clusters, a highly effective solution exists that is both very fast and almost non-redundant. A FORTRAN implementation is given in Program 1. Implementation details of this program can be found in Van Os (2001), as well as a hybrid approach that combines fast and slow-but-precise implementations. Recentely, a vectorized implementation of the fast approach is proposed that speeds-up relatively large problems ($N > 20$) up to 5 times by exploiting memory caches (submitted manuscript).

2.7 Resources Needed by the Algorithm

The resources needed for providing an exact solution are considerable. Although the calculation of the exact number of evaluations needed is cumbersome, it can be numerically verified that they all have the property that the workload approximately grows with a factor three if the object set size is increased by one. The order of the algorithm therefore is $O(3^N)$. In Figure 1 the development of workload for the most efficient process is plotted for partitioning a fixed number of objects N into an increasing number of clusters K. The figure shows a remarkable similarity between problems having a different number of objects. In general, partitioning into two clusters is a special case of smaller magnitude. As mentioned before, the DP workload is equivalent here to complete enumeration and equals 2^N. On the other hand, partitioning into three clusters does require a huge workload, and this continues to be the case up to partitioning into roughly $N/3$ clusters. The development of actually running time on a computer follows this pattern, with an increase factor between 2.7 and 3.7 for each extra object added, depending upon the amount of memory cache installed. For problems where the storage requirements approach or exceed the physical memory available, the running time increases more due to memory swapping.

The storage facilities needed for different problems are of order $O(2^N)$. In Table 3 the requirements for different problems are summarized. The requirements for ‘optimization only’ (first column) refer to the requirements for finding the objective function value for the optimal solution, as well as the last found cluster of that solution (the cluster that includes the object with the highest index). The second column gives requirements for retrieving the complete optimal cluster solution, including all clusters, while the third column gives storage requirements for retaining all information of the complete DP process, such that for example optimal solutions to subproblems can be retrieved. As a result of these requirements, it can be concluded that the maximum problem size to be handled by modern desktop computers is in the range of 25-29

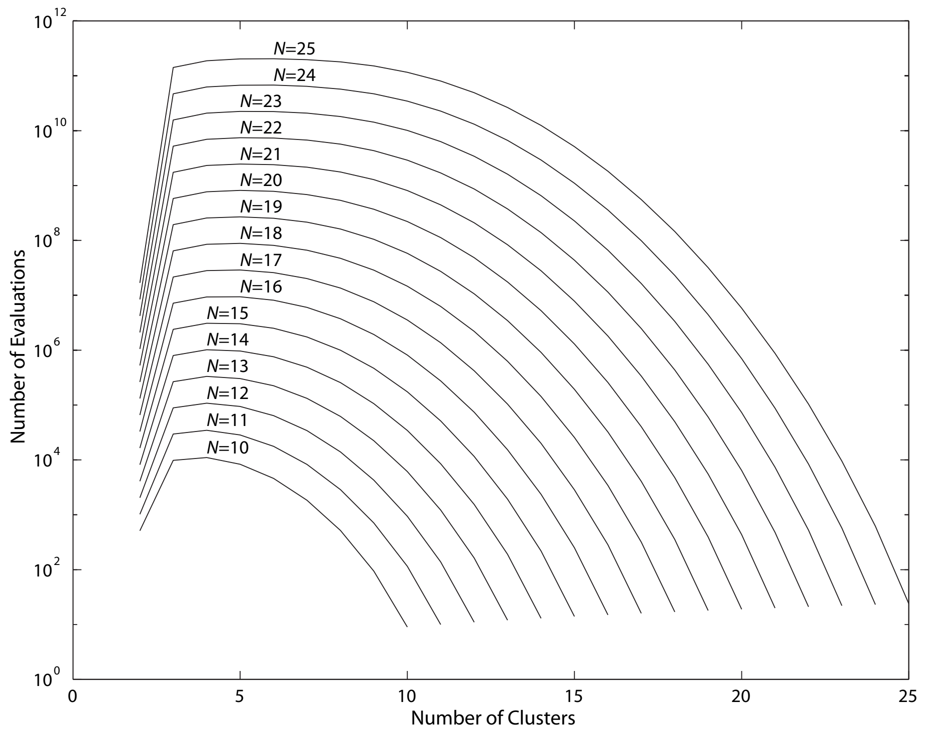

Figure 1: Number of evaluations needed by DP for partitioning N objects.

objects. If the available memory is limited, the requirements of all problems can be reduced to that of the least demanding problem (the last cell in first column), by solving the problem while retrieving only the ‘last found’ cluster, and afterwards retrieving the other clusters by solving reduced problems. This trades memory requirement for (often very) little additional execution time, because the reduced problems require only $\frac{1}{3}^{N-|S_K|}$ times the execution time of the unreduced problems.

2.8 Applications

The DPP algorithm can be applied to many different problems in clustering of objects and variables (e.g., see Hubert et al. 2001; Van Os 2001). Here we apply the algorithm first to the prototypical case of an optimization approach to clustering: K-means clustering, or minimizing the sum of squared error (sum of squared distances) between an object and its cluster centroid. Apart from the classical heuristic (relocation) algorithms that do not guarantee an optimal solution, one particular exact polynomial algorithm has been proposed by Hansen,

Table 3: Storage requirements for optimal speed implementations of DP, partitioning N objects into K clusters.

Clusters	Optimization Only	Optimization + Partitioning	Full Information
$2, \ldots, K$	$\frac{3}{2}2^N$	$(\frac{1}{2}K+1)(2^N)$	$K2^N$
K	$\frac{3}{4}2^N$	$\frac{7}{8}2^{N+1}$	$\frac{9}{8}2^{N+1}$

The storage requirement is the total number of integers and single precision real numbers to be stored simultaneously.

Jaumard, and Mladenovic (1998) for *bipartitioning* (i.e. $K = 2$) and hierarchical devisive clustering particularly for this criterion, which can deal with fairly large problems in low dimensional space. In contrast, the current algorithm finds *partitions* (i.e. $K \geq 2$) for relatively small problems without restrictions on the dimensionality. In the last application the algorithm will be applied to cluster problems that explicitly operate on a dissimilarity matrix. The data of the first application are originally from Euromonitor (1979, pp. 76-77), taken from Hand, Daly, Lunn, McConway, and Ostrowski (1994) (the data are also available at `http://lib.stat.cmu.edu/DASL/Datafiles/EuropeanJobs.html`). The data consist of the percentage employed in nine different industries in 26 European countries during the Cold War. The nine industries are: agriculture, mining, manufacturing, power supplies, construction, service industries, finance, social and personal services, and transport and communications.

As is generally the case in the optimization approach, the algorithm does not provide an immediate solution for determining the number of clusters needed. However, besides the use of an external criterion for the number of clusters (for an overview, see Milligan and Cooper 1985; Milligan 1996) that suggest a three-cluster solution for the present analysis, the DPP algorithm can be used to provide a series of optimal solutions for a number of clusters. Given the optimality guarantee, such a series of solutions will give the user an exact account of the usual trade-off between a greater complexity of the solution (larger number of clusters) and a larger variance accounted for (VAF). In this application, the algorithm has been used to provide the exact solutions for clustering in two up to nine groups. On a PC with a P4 3.1 GHZ chip it took 1.4 GB memory and approximately 14 hours to get these nine solutions. (The HAM program was not actually run on this problem, but would have taken over 450 days and 4.5 GB of memory.)

In Table 4 partitions from two up to seven clusters are given.

Table 4: Partitioning European Countries upon Employment in 9 Industries

	Number of clusters					
Countries	2	3	4	5	6	7
Belgium	1	1	1	1	1	1
Denmark	1	1	1	1	1	1
Netherlands	1	1	1	1	1	1
Finland	1	1	1	1	1	1
Norway	1	1	1	1	1	1
Sweden	1	1	1	1	1	1
Ireland	1	1	1	1	2	2
United Kingdom	1	1	1	1	2	2
France	1	1	1	2	2	2
W. Germany	1	1	1	2	2	2
Luxembourg	1	1	1	2	2	2
Austria	1	1	1	2	2	2
Italy	1	1	1	2	3	3
Switzerland	1	1	1	2	3	4
Spain	1	1	2	2	3	4
Portugal	1	1	2	3	3	3
Greece	1	2	2	3	4	3
Bulgaria	1	2	2	3	4	5
Poland	1	2	2	3	4	5
Rumania	1	2	2	3	4	5
USSR	1	2	2	3	4	5
Czechoslovakia	1	2	3	4	5	6
E. Germany	1	2	3	4	5	6
Hungary	1	2	3	4	5	6
Turkey	2	3	4	5	6	7
Yugoslavia	2	3	4	5	6	7
VAF	27%	48%	57%	64%	68%	72%

The three-cluster solution makes a major split between the West European countries and the Eastern bloc countries. Turkey and Yugoslavia (the latter still one country at the time) form a cluster very distinct from the others. The six-cluster solution splits the western countries into three groups, and separates the Eastern bloc countries Czechoslovakia, East Germany and Hungary from the rest, a grouping persistent among other partitions. Greece, Portugal, Spain and Switzerland are countries that share characteristics of several groups and are therefore not consistently assigned to any of these groups, depending on the number of clusters taken.

The data for the second application describe the protein consumption of 25 European countries (Turkey is not included) for nine food groups (Weber 1973, taken from Hand et al. 1994): red meat, white meat, eggs, milk, fish, ce-

Table 5: Partitioning European Countries based on Protein consumption

	Number of clusters					
Countries	2	3	4	5	6	7
Albania	1	1	1	1	1	1
Bulgaria	1	1	1	1	1	1
Romania	1	1	1	1	1	1
Yugoslavia	1	1	1	1	1	1
Hungary	1	1	1	2	2	2
USSR	1	1	1	2	2	2
Czechoslova	2	2	2	2	2	2
Poland	2	2	2	2	2	2
E Germany	2	2	2	2	2	3
Austria	2	2	2	3	3	3
Netherlands	2	2	2	3	3	3
W Germany	2	2	2	3	3	3
Belgium	2	2	2	3	3	4
France	2	2	2	3	3	4
Ireland	2	2	2	3	3	4
Switzerland	2	2	2	3	3	4
UK	2	2	2	3	3	4
Denmark	2	2	3	4	4	5
Finland	2	2	3	4	4	5
Norway	2	2	3	4	4	5
Sweden	2	2	3	4	4	5
Greece	1	3	4	5	5	6
Italy	1	3	4	5	5	6
Portugal	1	3	4	5	6	7
Spain	1	3	4	5	6	7
VAF	38%	51%	60%	66%	72%	77%

reals, starchy foods, nuts (pulses, nuts, and oil-seeds), and fruits and vegetables. The DPP-algorithm was used to provide optimal partitions up to nine clusters, which took 4.3 hours (P4 3.1 GHZ microcumputer) and 320 MB physical memory. Table 5 gives the partitions from two up to seven clusters.

The three-cluster partition splits the countries in a North/West/Central European cluster that eats more red and white meat, eggs and milk; four South European countries that eat much more fish, fruit and vegetables; and six former Eastern bloc countries that eat more cereals and nuts, and little fish and dairy products. All six partitions show a distinct grouping of countries in geographical regions that exhibits a largely hierarchical nature, where, for example, in a five-group partition, the Central/West European countries are separated from Scandinavia that eats considerably less meat, very little fruit, vegetables and nuts, and much more fish. North/mid-East European countries are split from

South-east European countries that have a distinct pattern of extremely little fish and starchy foods consumption, little white meat, eggs and milk and a very large consumption of nuts and especially cereals (the corresponding geographical partition is displayed in Figure 2).

The third application applies the algorithm to data involving inter-occupation dissimilarities. The data are derived by Smith (2001) from a subset of the U.S. Department of Labor Employment and Training Administration's O*NET Content Model Data (1998, pp. 739–742) which contains mean ratings on the level of 48 occupational skills associated with 25 occupational units. Thus, the data to be analyzed consist of Euclidean distances among these 25 occupations: accountant, photographer, optometrist, engineer, architect, farmer, electrician, actuary, technical writer, reporter, school administrator, social worker, dietician, chef, lawyer, realtor, biologist, chemist, computer programmer, physical therapist, nurse R.N., physicist, dentist, carpenter, and musician. To demonstrate the flexibility of the algorithm, the dissimilarity data were clustered optimally in four different ways by *solving* Mathematical Program 1 for four different homogeneity functions $h(S_k)$ defined on each cluster S_k. These are (a) the sum of dissimilarities in S_k; (b) the sum of dissimilarities in S_k divided by the number of objects in S_k; (c) the number of instances in which the dissimilarity within S_k is strictly greater than one between S_k and $S - S_k$ (dissimilarity inconsistencies); and (d) the number of times an object within S_k has a larger dissimilarity to another object within S_k than it has to an object in $S - S_k$ (object inconsistencies). (The average dissimilarity in S_k was also considered, but solutions based on this criterion show a strong tendency to include as many singleton clusters within each partition as possible, typically $K - 1$ singletons, and is therefore hereafter not considered.)

The DPP-algorithm was used to provide optimal partitions up to eight clusters for all four criteria, which took 14.9 hours (P4 3.1 GHZ microcumputer) and 640 MB virtual memory. The sequences of optimal partitions with increasing number of clusters for criteria a, c, and d exhibit a largely hierarchical nature, similar to the hierarchies found by Smith (2001), who fitted ultrametric and additive trees based on a L_1-norm. The partitions obtained for criteria b, c, and d are strongly similar, and the four- and five- cluster partitions for criteria b, c, and d are equivalent. Criterion a, that minimizes the sum of dissimilarities in S_k, resulted in partitions where all clusters appear to have equal size as much as possible, and these partitions do not exhibit a hierarchical nature.

To determine the number of clusters two indices were computed for all optimal partitions found: the C-Index (Hubert and Levin 1976) that is based upon the sum of within cluster dissimilarities (function a.); and the Gamma-Index (Baker and Hubert 1975) that is based upon the number of consistent/

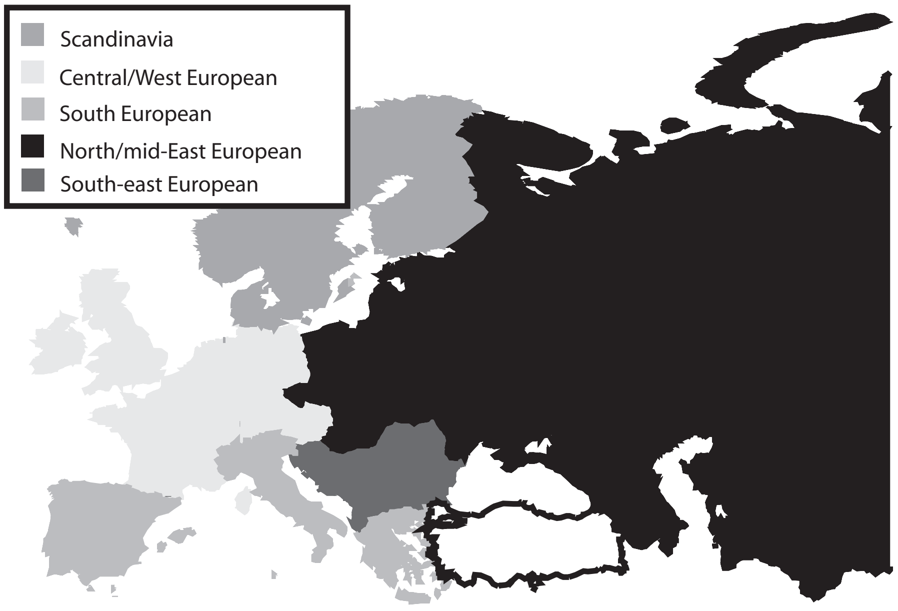

Figure 2: A geographical display of a partition of Europe based upon protein consumption.

inconsistent comparisons involving between and within cluster distances (function c.) The number of clusters of the partitions with lowest C-Index and the highest Gamma-Index indicate the ‘correct’ number of clusters. When examining these index values as a function of an increasing number of clusters for each particular criterion, both a noticeable decrease in the C-Index values as well as an increase in Gamma values indicated that at least four clusters were needed. The optimal index values occur at a rather high number of clusters, indicating partitions with too many clusters to be informative. The differences, however, between those optimal index values and the index values for partitions with seven clusters are small, and both indices are known to have a slight tendency to overestimate the number of clusters (Milligan and Cooper 1985). Therefore, a maximum of seven clusters was chosen. In Table 6 optimal partitions into seven clusters for all four criteria are given.

Several clusters can be readily identified, such as a cluster of techni-

Table 6: Partitions into 7 clusters of 25 Occupations according to overall skill dissimilarities for four objective functions.

Objective function			
sum of dissimilarities	weighted sum of dis.	dis. inconsistencies	object inconsistencies
actuary	actuary	actuary	actuary
technical writer	technical writer	technical writer	technical writer
reporter	reporter	reporter	reporter
architect	architect	architect	architect
school administrator	school administrator	school administrator	school administrator
dietician	dietician	dietician	dietician
accountant	accountant	accountant	accountant
lawyer	lawyer	lawyer	lawyer
realtor	realtor	realtor	realtor
social worker	social worker	social worker	social worker
optometrist	optometrist	optometrist	optometrist
nurse R.N.	nurse R.N.	nurse R.N.	nurse R.N.
dentist	dentist	dentist	dentist
physical therapist	physical therapist	physical therapist	physical therapist
electricia	electrician	electrician	electrician
carpenter	carpenter	carpenter	carpenter
photographer	photographer	photographer	photographer
farmer	farmer	farmer	farmer
chef	chef	chef	chef
musician	musician	musician	musician
computer programmer	computer programmer	computer programmer	computer programmer
engineer	engineer	engineer	engineer
biologist	biologist	biologist	biologist
chemist	chemist	chemist	chemist
physicist	physicist	physicist	physicist

cal/scientific occupations (engineer, biologist, chemist, physicist, programmer); a ‘health-service’ cluster (optometrist, nurse R.N., dentist, physical therapist) that might be combined with a cluster involving communication skills (realtor, social worker), or if separated, includes the lawyer; a ‘writing’ cluster (actuary, technical writer, reporter); a ‘technical hand-tool’ cluster (electrician, carpenter). Two clusters are more difficult to interpret, the rather heterogeneous cluster of architect, school administrator, dietician, and accountant); and an ‘artistic skill’ cluster (photographer, chef, musician) that oddly includes ‘farmer’. This might be explained by the fact that the occupation of farmer also involves a rather heavy use of tools, and in the five-cluster solution these occupations are consistently combined with the other ‘tool’ cluster (electrician, carpenter).

To indicate the difficulty of obtaining optimal solutions by a heuristic strategy for these data, all problems were also ‘solved’ by an iterative relocation heuristic that executes the best object to centroid relocation available at each iteration and updates the centroids immediately (see, Van Os 2001, p. 24), a K-means variant comparable to those proposed by Späth (1985) and Hanson

Table 7: The proportion of (local) optima out of 100 starts found by an iterative relocation algorithm for the occupations data.

	Objective Function			
K	a	b	c	d
2	.21	.73	.50	.29
3	.03	.83	.99	.01
4	.40	.78	.94	.66
5	.08	.55	.97	.56
6	.06	.09	.31	.39
7	**.01**	.14	.22	.20
8	**.01**	.03	.10	.13
9	.02	.04	**.04**	.05
10	**.01**	.06	**.18**	.06
11	**.01**	.09	**.18**	.06
12	**.01**	.11	**.28**	.03

Bold values indicate a local optimum

and Mladenovic (2001). Using 100 random starts, for each problem the best solution out of those 100 starts was compared to the optimal solution found by the DP algorithm for the same problem, and the number of times this (local) optimum was found was determined. The results are displayed in Table 7. It appears that the occurrence of local optima increases as the number of clusters increases, and is especially severe for the sum of dissimilarities criterion (a), for which the global optimum is almost never found.

2.9 Discussion

The present paper gives a much more efficient way of exploiting the DP partitioning recursion. At the same time the application of the algorithm shows that a high price in terms of computing resources is to be paid for obtaining an exact solution. Often, for well structured data, good heuristics can provide the same solutions in seconds or minutes, but without the guarantee of optimality. For other data sets and more difficult objective functions, many local optima may occur (for examples, see Van Os 2001). Moreover, given the current low cost of computing resources and the effort and time that is usually needed for collecting the data, for many scientific purposes the resources needed may be considered relatively minor. The main limitation of the algorithm therefore is in its practical inability to handle data sets larger than, say, 28 objects. Although computing resources increase continuously, it is not expected that this limit will be extended greatly in the near future given the order $O(3^N)$ of the algorithm; this is however to be expected given the NP-hardness of the general partitioning problem. As an alternative, the exact DP algorithm has been extended into a heuristic approach that can handle large datasets, although no longer with a guaranteed global optimum (Hubert et al. 2001), and that combined with an approach similar to genetic algorithms appears to be fairly effective (see Van Os 2001). The latter source also shows how the DP algorithm can be accelerated further for specific cases by combining branch-and-bound techniques with DP, but this does not affect the storage requirements needed and therefore does not alter the general size limit.

The attractiveness of DP as an exact algorithm certainly stems from its flexibility to handle many different optimization approaches to clustering (i.e., many different objective functions). The overview given by Hansen and Jaumard (1997) provides several alternative exact algorithms but they all handle very specific cases; none of these approaches have the general applicability of the current algorithm. Also, the resources needed by our algorithm are known in advance, whereas (mixed-)integer linear programming (ILP) techniques provide fast solutions for some data sets, but seem to run forever when applied to others. Indeed, as is suggested by a comparative study of a DP algorithm and a ILP algorithm for seriation (Brusco 2001), DP may be the method of choice for problems that are within its inherent size limits.

The applicability of our algorithm therefore spans into several areas: the exact analysis of small data sets, the approximate analysis of large data sets through heuristic extensions, and its ability to provide benchmarks for verifying the ability of heuristics to provide optimal solutions in comparative studies. The latter also gives a more thorough understanding of the relative merits of different objective functions. Historically, comparative studies of cluster methods (for an overview, see Milligan 1996) are subject to the fact that the many cluster methods proposed involve the combination of *both* an algorithm and a cluster criterion. Studying the many cluster criteria/objective functions in combination with modern heuristics may reveal that some objective functions are indeed effective but much more difficult to apply optimally, as is suggested in Van Os (2001, pp. 103–105) for objective functions designed for clustering variables based upon correlations and canonical correlations.

2.10 Appendix: A Fortran Program for Single Stage Recursion Processing

Program 1 A single stage of the recursion

```
      SUBROUTINE DPPSingleStage (F1,Fk_1,Fk,NFk,N,MOpt,K,
     +                          NSub,LastOnly)
C Function:     Processes stage k (2<k<NSub) of the recursion of
C               DP partitioning
C Version:      1.0, Januari 2001
C Author:       B. J. van Os
C     N:        Number of objects
C     NSub:     Number of clusters
C     Mopt:     (1=MIN SUM, 2=MIN MAX)
C     LastOnly:  (.TRUE. = process for solution in NSub clusters only,
C               .FALSE.= process for solutions in 2..NSub clusters)
C     Fk:       Fk(AB) is optimal F for partitioning set AB at stage k
C     Fk_1:     Fk_1(AB) is optimal F for partitioning set AB at stage k-1
C     F1:       F1(A) is heterogeneity of set A (first stage)
C               if F1(A)=Z, A is considered not admissable (bounding)
C     NFk:      NFk(AB) last added cluster to optimal partition of AB
C     Size:     function that returns the number of objects (# bits set)
C               of its argument
C Parameter block.
      INTEGER     N,MOpt,K,NSub
      REAL        F1((2**N)-1),Fk_1((2**N)-1),Fk((2**N)-1)
      INTEGER     NFk((2**N)-1)
      LOGICAL     LastOnly
C Externals
      INTEGER     Size
C Local variables/constants
      REAL        Z
      PARAMETER   (Z=1.0E+20) ! sufficiently large
      INTEGER     FirstA,O,LastA,MaxO
      INTEGER     IndexAB,IndexA,IndexB,LastB,IncrB,NotA
      REAL        Temp
```

Program 1 continued

```
C Program start.
      MaxSizeAB=(N-(NSub-K))
      IF (LastOnly) THEN
        MaxO=MaxSizeAB
      ELSE
        MaxO=N-1
      END IF

C     O is the leftmost object to be 'taken out'  out of the
C     (N-(NSub-K)) right objects. Loop over all admissable O
      DO O=MaxO,K,-1
         MaxSizeA = Min ((1+O-K),N-(NSub-1))
         FirstA=ISHFT(1,O-1)
         LastA=ISHFT(1,O)-1
C        initialize F for sets AB (when overwriting Fk_1 with Fk)
         DO IndexAB = FirstA,LastA
            Fk(IndexAB)  = Z
         END DO

C        in IndexA all possible sets A will be generated
C        out of the O rightmost objects that at least include object O
         DO IndexA = FirstA,LastA
            IF ((Size(IndexA) .LE. MaxSizeA)
     +      .AND. (F1(IndexA) .LT. Z)) THEN
C              Given admissable IndexA, in IndexB all possible subsets
C              out of the O rightmost objects have to be generated,
C              that do not include IndexA, to form IndexAB=IndexA+IndexB
               NotA=NOT(IndexA)
               IndexB=IEOR((ISHFT(1,O)-1),IndexA)
               LOOPB: DO WHILE (IndexB .GT. 0)
                  IndexAB=IndexA+IndexB
                  IF (MOpt .EQ. 1) THEN
                     Temp=Fk_1(IndexB)+F1(IndexA)
                  ELSE
                     Temp=MAX(Fk_1(IndexB),F1(IndexA))
                  END IF
                  IF(Fk(IndexAB).GE.Temp) THEN
                     Fk(IndexAB)  = Temp
                     NFk(IndexAB) = IndexA
                  END IF
                  IndexB = IAND((IndexB-1),NotA)
               END DO LOOPB
            END IF
         END DO ! over all A
      END DO ! over all O
      RETURN
      END
```

References

ALPERT, C. J., and KAHNG, A. B. (1995). “Multiway Partitioning via Geometry Embeddings, Orderings, and Dynamic Programming,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, *14*, 1342–1357.

ALPERT, C. J., and KAHNG, A. B. (1997). “Splitting an Ordering into a Partititon to Minimize Diameter,” *Journal of Classification*, *14*, 51–74.

ARABIE, P., HUBERT, L. J., and DE SOETE, G.(1996). *Clustering and Classification*, Singapore: World Scientific Publishing.

BAKER, F. B., and HUBERT, L. J. (1975). “Measuring the Power of Hierarchical Cluster Analysis,” *Journal of the American Statistical Association*, *70*, 31–38.

BRUSCO, M. J. (2001). “Seriation of Asymmetric Matrices Using Integer Linear Programming.,” *British Journal of Mathematical and Statistical Psychology*, *54*, 367–375.

BRUSCO, M. J. (2003). “An Enhanced Branch-and-bound Algorithm for a Partitioning Problem.,” *British Journal of Mathematical and Statistical Psychology*, *56*, 83–92.

DAY, W. H. E. (1996). “Complexity Theory: An Introduction for Practitioners of Classification,” *Clustering and classification*, Eds., P. Arabie, L. J. Hubert, and G. De Soete, Singapore: World Scientific Publishing, 199–233.

DIEHR, G. (1985). “Evaluation of a Branch and Bound Algorithm for Clustering,” *SIAM Journal on Scientific and Statistical Computing*, *6*, 268–284.

DODGE, Y., and GAFNER, T. (1994). “Complexity Relaxation of Dynamic Programming for Cluster Analysis,” *New Approaches in Classification and Data Analysis*, Eds., E. Diday, Y. Lechevallier, M. Schader, P. Bertrand, and B. Burtschy, Berlin: Springer, 220–227.

EUROMONITOR, (1979). *European Marketing Data and Statistics*, London: Euromonitor Publications, (also see http://lib.stat.cmu.edu/DASL/Datafiles/EuropeanJobs.html)

FISHER, W. D. (1958). “On Grouping for Maximum Heterogeneity,” *Journal of the American Statistical Association*, *59*, 789–798.

HAND, D. J., DALY, F., LUNN, A. D., MCCONWAY, K. J., and OSTROWSKI, E. (1994). *A Handbook of Small Data Sets*, London: Chapman & Hall.

HANSEN, P., and DELATTRE, M. (1978). “Complete-link Cluster Analysis by Graph Coloring,” *Journal of The American Statistical Association*, *73*, 397–403.

HANSEN, P., and JAUMARD, B. (1997). “Cluster Analysis and Mathematical Programming,” *Mathematical Programming*, *79*, 191–215.

HANSEN, P., JAUMARD, B., and MLADENOVIC, N. (1998). “Minimum Sum of Squares Clustering in a Low Dimensional Space,” *Journal of Classification*, *15*, 37–56.

HANSON, P., and MLADENOVIC, N. (2001). “J-means: A New Local Search Heuristic for Minimizing Sum of Squares Clustering.,” *Pattern Recognition*, *34*, 405–413.

HARTIGAN, J. A. (1975). *Clustering Algorithms*, New York: Wiley.

HUBERT, L. J., ARABIE, P., and MEULMAN, J. J. (2001). *Combinatorial Data Analysis: Optimization by Dynamic Programming*, Philadelphia: SIAM.

HUBERT, L. J., and LEVIN, J. R. (1976). “A General Statistical Framework for Assessing Categorical Clustering in Free Recall,” *Psychological Bulletin*, *83*, 1072–1080.

JENSEN, R. E. (1969). “A Dynamic Programming Algorithm for Cluster Analysis,” *Journal of the Operations Research Society of America*, *7*, 1034–1057.

KLEIN, G., and ARONSON, J. E. (1991). “Optimal Clustering: A Model and Method,” *Naval Research Logistics*, *38*, 447–461.

KOONTZ, W. L. G., NARENDRA, P. M., and FUKUNAGA, K. (1975). “A Branch and Bound Clustering Algorithm,” *IEEE Transactions on Computers*, *C–24*, 908–915.

LIM, K., and LEE, Y.-H. (1997). “Optimal Partitioning of Heterogeneous Traffic Sources in Mobile Communications Networks,” *IEEE Transactions on Computers*, *46*, 312–325.

MILLIGAN, G. W. (1996). “Clustering Validation: Results and Implications for Applied Analyses,” *Clustering and classification*, Eds., P. Arabie, L. J. Hubert, and G. De Soete, Singapore: World Scientific Publishing, 341–375.

MILLIGAN, G. W., and COOPER, M. C. (1985). “An Examination of Procedures for Determining the Number of Clusters in a Data Set.,” *Psychometrika*, *50*, 159–179.

NIJENHUIS, A., and WILF, H. S. (1975). *Combinatorial Algorithms*, New York: Academic Press.

OLSTAD, B., and MANNE, F. (1995). “Efficient Partitioning of Sequences,” *IEEE Transactions on Computers*, *44*, 1322–1326.

SMITH, T. J. (2001). “Constructing Ultrametric and Additive Trees Based on the l_1 norm,” *Journal of Classification*, *18*, 185–207.

SPÄTH, H. (1985). *Cluster Dissection and Analysis (Theory, Fortran Programs, Examples)*, Chichestor: Ellis Horwood.

U.S. Department of Labor Employment and Training Administration, (1998). *O*net 98 Data Dictionary*, Raleigh, NC: Trefoil Corporation.

VAN OS, B. J., (2001). *Dynamic Programming for Partitioning in Multivariate Data Analysis*, Unpublished doctoral dissertation, Leiden University.

WEBER, A. (1973). *Agrarpolitik im Spannungsfeld der Internationalen Ernaehrungspolitik*, Kiel: Institut fuer Agrarpolitik und marktlehre.