

Fast Clustering Algorithms

ULRICH DORNDORF / *INFORM—Institut für Operations Research und Management GmbH, Pascalstraße 23, D-52076 Aachen, F.R.G., Email: uli%inform.uucp@germany.eu.net*

ERWIN PESCH / *Faculty of Economics and Business Administration, KE, University of Limburg, P. O. Box 616, NL-6200 MD Maastricht, The Netherlands, Email: e.pesch@ke.rulimburg.nl*

(Received: February 1992; final revision received: March 1993; accepted: May 1993)

This paper considers the problem of partitioning the vertices of a weighted complete graph into cliques of unbounded size and number, such that the sum of the edge weights of all cliques is maximized. The problem is known as the clique-partitioning problem and arises as a clustering problem in qualitative data analysis. A simple greedy algorithm is extended to an ejection chain heuristic leading to optimal solutions in all practical test problems known from literature. The heuristic is used to compute an initial lower bound as well as to guide branching in a branch and bound algorithm which is superior to present exact methods. Empirical data for all three algorithms are reported.

A task that frequently arises in qualitative data analysis is to uncover natural groupings, or types, of objects, each of which is characterized by several attributes. One can think of these objects as vertices of an edge-weighted graph, G ; each positive or negative weight represents some measure of similarity or dissimilarity, respectively, of an edge-defining object pair. A clustering of the objects into groups is a partition of the graph, which means a partition of the vertex set of G into non-overlapping subsets. The set of edges connecting vertices of different subsets from some partition of G is called a cut. In order to find groups as homogeneous as possible, positive edges should appear within groups and negative edges in the cut. Hence, a best clustering is one with a minimal cut weight.

Cut minimization subject to some additional constraints arises in many applications, and the literature straddles all quantitative, scientific disciplines, as demonstrated by the remarkable variety in the reference section of Dubes and Jain.^[11]

In VLSI (Very Large Scale Integration) components of an electronic circuit have to be placed on a silicon wafer. The components communicate with each other through wires. Hence, grouping highly connected components will meet two basic objectives, minimization of the total wire length and minimization of the area required for placing the components of the chip (see Kernighan and Lin,^[35] Dunlop and Kernighan,^[12] Lengauer,^[38] and Feo and Khellaf^[14]).

A computer program may be thought of as a set of blocks, for instance, subroutines or procedures. Connections, like procedure calls, between these blocks occur via

data or control flows. In paged memory organizations these blocks have to be assigned to pages of a fixed size such that the number of calls between blocks of different pages is minimized (see Kernighan^[34] and the remarks on compiler construction in Johnson et al.^[31]).

There are other applications from economics like grouping in flexible manufacturing systems (cf. Kumar et al.^[37] or Ahmadi and Tang^[1]), from biology, from political and social science, etc., which can all be translated into some type of a graph partitioning problem (see, for instance, Benders^[7] or Barthélemy and Monjardet^[6]). Figure 1 summarizes several graph partitioning problems; for each problem beside its name there is a brief description as well as some references and a remark on its complexity. In all of these problems the vertex set V of the underlying graph is partitioned into k subsets of a possibly predefined size. Hence, the number of vertices $|V_i|$ in subset i is bounded by some lower bound l_i and some upper bound u_i . Edge weights may be positive or negative. The objective function is to minimize—to maximize in case of the max-cut problem—the external costs, i.e., the sum of the edge weights of all edges connecting vertices of different subsets.

Linear and non-linear binary problem formulations (see Lukes,^[40] Barnes et al.,^[5] Kumar et al.,^[37] Roucairol and Hansen^[49]) and set partitioning problem formulations (see Minoux and Pinson^[44]) led to the development of sophisticated exact and approximation algorithms. We refer the reader to the branch and bound or branch and cut algorithms from Roucairol and Hansen,^[49] Grötschel and Wakabayashi,^[24–26] Chopra and Rao,^[9] Deza et al.,^[10] the column generation method of Johnson et al.,^[31] or a dynamic programming approach from Lukes.^[40] The most prominent representative among the approximation algorithms is the algorithm of Kernighan and Lin,^[35] the predecessor of the famous paper of Lin and Kernighan^[39] on the traveling salesman problem. Modifications of the Kernighan-Lin algorithm are used in Fiduccia and Mattheyses,^[15] Krishnamurthy,^[36] and Bui et al.^[8] For other local search approaches based on simulated annealing (Johnson et al.^[30] and Amorim et al.^[3]), tabu search (Amorim et al.^[3]), genetic algorithm hybrids (Mühlenbein et al.^[45]) see the references.

Subject classifications: Data Analysis. Clustering.

Other key words: Local search and branch and bound algorithms for clique partitioning, ejection chains, propagation of constraints.

min-cut	$k = 2$; easy; Ford and Fulkerson ⁽¹⁶⁾
k -way-partitioning	$k > 2$; NP-complete; Chopra and Rao ⁽⁹⁾ , Minoux and Pinson ⁽⁴⁴⁾
k -partitioning	$k \geq 2$; $\sum_1^k u_i$ for $i = 1, \dots, k$; NP-complete; Garey, Johnson, and Stockmeyer ⁽¹⁸⁾ , Garey and Johnson ⁽¹⁷⁾
clique-partitioning	k not fixed; NP-complete for positive and negative edge weights; Wakabayashi ⁽⁵⁴⁾ , Grötschel and Wakabayashi ⁽²⁴⁾ (25) (26)
m -dimensional matching	$k \geq 2$; $\sum_1^k u_i = m$ for all $i = 1, \dots, k$; NP-complete; easy for $m = 2$; Feo and Khellaf ⁽¹⁴⁾ , Papadimitriou and Steiglitz ⁽⁴⁷⁾
max-cut	$k = 2$; NP-complete; Garey, Johnson, and Stockmeyer ⁽¹⁸⁾
hypergraph partitioning	Sanchis ⁽⁵⁰⁾ , Kahng ⁽³³⁾

Figure 1. Graph partitioning problems.

More about data analysis and clustering can be found in the books of Hand,^[28] Hartigan,^[29] Anderberg,^[4] and Späth.^[52]

1. The Clique-Partitioning Problem

The objective of cluster analysis is to uncover natural groupings, or types, of n objects, each of which is characterized by m attributes. Describe in some sense the best partitioning of all objects into disjoint subsets or classes (also called clusters) such that each cluster is as homogeneous as possible with respect to all attributes. We can think of all kinds of objects such as animals, countries, computers, markets, companies, etc., as we will see later. The set of n objects may be described by some $n \times m$ matrix $D = (d_{ij})$, where each entry d_{ij} , $i = 1, \dots, n$ and $j = 1, \dots, m$, represents the property of object i with respect to attribute j . In general there are two steps to be performed during the clustering process. First, there must be some measure of similarity between distinct objects, and second, according to these similarities the objects must be clustered into groups. Only the last step corresponds to the clique-partitioning problem. To find some measure of similarity for qualitative data we can proceed in the following way (see, for instance, Hartigan,^[29] Späth,^[52] or Grötschel and Wakabayashi^[24]). For two objects i and j let \tilde{m} be the number of attributes k of the same value (property), i.e., where $d_{ik} = d_{jk}$. Then some measure of similarity is the value $w_{ij} := 2 \cdot \tilde{m} - m$, where $w_{ij} = -m$ says that both objects are completely different while increasing values w_{ij} correspond to increasing similarity of the objects. The objects may be considered as vertices of an undirected complete graph with edge weights w_{ij} where

$$w_{ij} = 2 \cdot |\{k | d_{ik} = d_{jk}, k = 1, \dots, m\}| - m, \quad \text{for all } 1 \leq i < j \leq n. \quad (1)$$

Hence, clustering of the objects corresponds to partitioning the complete graph of n vertices into complete subgraphs, or cliques, such that the sum of all edge weights in all cliques is maximum, or equivalently

$$\min \sum_{1 \leq k < \ell \leq n} \sum_{i \in V_k, j \in V_\ell} w_{ij}$$

where V_k and V_ℓ are the vertex sets of cliques k and ℓ , respectively.

Obviously, if all edge weights are nonnegative or non-positive the problem can easily be solved. If the graph has negative as well as positive edge weights the clique-partitioning problem is NP-complete (cf. Wakabayashi,^[54] Dyer and Frieze^[13]).

If we assign to each edge (i, j) a binary variable x_{ij} where

$$x_{ij} = \begin{cases} 1 & \text{if vertices } i \text{ and } j \text{ are in the same clique,} \\ 0 & \text{otherwise,} \end{cases}$$

then the following model gives a complete description of the clique-partitioning problem (see Grötschel and Wakabayashi,^[24, 25] Marcotorchino and Michaud,^[41-43] Opitz and Schader,^[46] Schader and Tüshaus,^[51] and Tüshaus^[53]).

$$\begin{aligned} \max \quad & \sum_{1 \leq i < j \leq n} w_{ij} \cdot x_{ij} \\ \text{s.t.} \quad & x_{ij} + x_{jk} - x_{ik} \leq 1 \quad \text{for } 1 \leq i < j < k \leq n \\ & x_{ij} - x_{jk} + x_{ik} \leq 1 \quad \text{for } 1 \leq i < j < k \leq n \\ & -x_{ij} + x_{jk} + x_{ik} \leq 1 \quad \text{for } 1 \leq i < j < k \leq n \\ & x_{ij} \in \{0, 1\} \quad \text{for } 1 \leq i < j \leq n. \end{aligned}$$

The constraints assure that, for each triangle (a clique of only three vertices) in the graph, if two edges belong to the same clique then the whole triangle belongs to this clique. For qualitative data analysis, the definition of the edge weights as mentioned before can easily be derived if the clusters are considered as equivalence classes of an equivalence relation whose m projections to each of the m attributes (or equivalence classes) lead to a maximum of coincidence (see Grötschel and Wakabayashi^[24] or Zahn^[55]). However, if quantitative attributes have to be considered, or in case of weighted or missing attributes, other edge weight computation formulas apply. For instance, in case of the micro problem which is mentioned below the crite-

riterion $d_{ik} = d_{jk}$ is replaced by the more general similarity criterion

$$\frac{|d_{ik} - d_{jk}|}{\max\{d_{ik}, d_{jk}, 1\}} \leq \alpha,$$

where $\alpha = 0.3$. In case of the classification of cetacea (dolphins, porpoises, and whales), which is also mentioned below, some information is missing (entry “*” in the data matrix). Therefore the weights w_{ij} are calculated such that only comparable attributes (no entry “*”) of the objects i and j are considered. Hence,

$$w_{ij} = 2 \cdot |\{k | d_{ik} = d_{jk}, d_{ik} \neq *, k = 1, \dots, m\}| - |\{k | d_{ik} \neq *, d_{jk} \neq *, k = 1, \dots, m\}|, \\ 1 \leq i < j \leq n. \quad (2)$$

The second term corresponds to the number of comparable attributes for both objects. If no information is missing then we get the usual definition of the edge weights.

The classification of 137 companies with respect to their need of different groups of employees (secretaries, programmers, engineers, etc.) seems to give a less interesting result. For 25 groups of employees, each company indicated whether the group is needed (entry “1”) or not (entry “0”). The optimal solution (see Table V) has two clusters, one containing only two companies (no. 10 and no. 107). Both companies mentioned require all kinds of employee groups. Hence, for a useful clustering weighted attributes seem to be indispensable, for instance, where the weighting factor for each attribute property is inversely proportional to the number of the attribute’s property occurrences over all objects (see Anderberg^[4]).

The latter weight function (2) is also used in our additional problem instance of classification of lecturers (see Appendix). The Association of Business Administration Lecturers from Germany, Austria, and Switzerland has 13 commissions, each of which covers a special subject such as marketing, banking and finance, operations research, production management, etc. Each member of the association is supposed to be a member of some commissions as well as to express interest in their subjects. To each lecturer we assigned a 0,1-tuple where an entry “1” indicates the membership in the corresponding commission, or the interest in its subject, respectively. Commission membership always implies interest in its subject but not vice versa. Therefore, a membership entry “1” yields also an interest entry “1”. If we use the first weight function (1) we get an optimal solution resulting in one cluster only. However, this is not a big surprise. For instance, if two lecturers have no common interest or are not members of a particular commission then the corresponding “0” entries do not provide some characteristic information that these two lecturers have in common. Entries of “1” give more information. Therefore, we used the second type of weight function (2) where a “0” entry in our data set was considered as “*”. All relevant data as well as the best solution we could

find, consisting of 114 clusters, is presented in the Appendix.

We are leaving aside the question whether a partition into disjoint clusters will be an appropriate setting for the companies and lecturers problem instances in order to yield meaningful interpretations. More information about the problem of a meaningful interpretation and aggregation of related attributes can be found in Grötschel and Wakabayashi,^[24] Opitz and Schader,^[46] or Anderberg.^[4]

2. An Ejection Chain Heuristic for the Clique-Partitioning Problem

We now present a powerful heuristic based on the ejection chain idea of Glover^[22,23] and the variable depth local search approach of Kernighan and Lin^[35] for the graph partitioning problem.

Consider a feasible solution X of the clique-partitioning problem, i.e., a partition of the vertices into a number of cliques. A solution X' is defined to be a neighbor of X if X' is obtained by the following operation (called a move): Move a vertex u from its current cluster in X to another perhaps non-existing and by vertex u newly defined cluster. Define the neighborhood of X to consist of all solutions X' obtained from X by one move, see Règnier.^[48] Obviously, the right sequence of moves will lead from any solution to an optimal solution of the clique-partitioning problem. In the simplest version of our local search algorithm we are always proceeding in a greedy way. Thus, each move to some neighbor of a feasible solution is always a most improving move, i.e., among all neighbors that yield an improved objective function value that one with the highest objective function value is chosen. Usually this process will end up in some suboptimal solution.

The same neighborhood structure was employed by Amorim et al.^[3] in their simulated annealing and tabu search approach.

Simple Local Search

In an n vertex graph the number of nonempty clusters is at most n . Starting from a random partition where each vertex randomly is assigned to one of the n clusters, some empty clusters may be left over. Let $v(u)$ be the label of the clique containing vertex u . Let $V_1, \dots, V_k, \dots, V_n$ be the clusters or cliques of the current solution where several of the V_i may be empty. Thus, vertex u is contained in clique $V_{v(u)}$. For each vertex u we can define internal costs $I(u)$ and external costs $E(u, k)$ where

$$I(u) = \sum_{\substack{i \in V_{v(u)} \\ i \neq u}} w_{ui} \quad \text{and} \quad E(u, k) = \sum_{\substack{i \in V_k \\ i \neq u}} w_{ui}$$

$I(u)$ is the sum of all edge weights of all edges connecting u and all other vertices in the clique containing vertex u . $E(u, k)$ is defined with respect to clique V_k . It is the sum of all edge weights of edges in the complete graph connecting u and all vertices in clique V_k . Obviously, $E(u, v(u)) = I(u)$. Each move of a vertex u from clique $V_{v(u)}$ to clique V_k yields the gain $g(u, k) = E(u, k) - I(u)$. Hence the best

move is that one for which $g(u, k)$ is maximum (and greater zero) for all vertices u and all cliques V_k . Let

$$g(u^*, k^*) + \max_{u \in V} \left\{ \max_{\substack{1 \leq k \leq n \\ k \neq v(u)}} \{g(u, k)\} \right\},$$

and let $E = (E(u, k))_{u=1, \dots, n; k=1, \dots, n}$ be an $n \times n$ matrix containing all external costs. If vertex u^* is moved from clique $V_{v(u^*)}$ to clique V_{k^*} , then the new entries $E(u, v(u^*))$ are $E(u, v(u^*)) - w_{u^*u}$ for all $u \in V \setminus \{u^*\}$. The entries $E(u, k^*)$ are changed to $E(u, k^*) + w_{u^*u}$ for all $u \in V \setminus \{u^*\}$. Hence, the number of elements changed is of order $\mathcal{O}(n)$. To find the best move, at most n^2 entries in the matrix E have to be considered. However, if we organize the entries of each row in E as a heap, the best move can be found within $\mathcal{O}(n)$ steps, and updating all heaps requires again $\mathcal{O}(n \cdot \log n)$ steps (cf. Aho et al.^[2]). Hence, the neighborhood can be explored with time complexity $\mathcal{O}(n \cdot \log n)$ instead of $\mathcal{O}(n^2)$ required by Amorim et al.^[3]

Ejection Chain Heuristic

The local search algorithm as described above might stop after a few iterations in some local optimum far away from the global optimum. To avoid this type of premature convergence, the idea of Kernighan and Lin^[35] may be adapted to the clique-partitioning problem. The algorithm is a special case of a more general approach introduced first by Glover.^[22, 23] The main idea is similar to the one presented in tabu search (see Glover [19–21]):

- (i) Greedily look for the best move which is not necessarily improving.
- (ii) Avoid cycling via a dynamically organized list of forbidden (tabu) moves.
- (iii) Perform an improving sequence of moves.

In case of the clique-partitioning problem we get the algorithm of Figure 2. Compressing a sequence of moves into a single compound move is a main idea of an ejection chain-based local search algorithm. Hence, the neighborhood for a simple type of move becomes embedded, level

by level, in successively larger neighborhoods that represent more complex moves (Glover^[22]). Contrary to our case, the component moves carried forward level by level need not in general yield feasible solutions. In order to avoid substantial infeasibility, each component move is usually forced by a component move on the preceding level, hence the name ejection chain. In case of the m -dimensional matching problem we would get a series of ejecting vertices, one by one, over the set of clusters, leading finally to a new feasible solution. For the clique partitioning problem we kept the ejection process fairly rudimentary because each component move never violates feasibility. Nevertheless, an implementation of Glover's idea of ejection chains can easily be achieved. The neighborhood of a component move in the chain will be much smaller than in our implementation, i.e., each move of a vertex to a new cluster ejects a new vertex from that cluster, and so on, until all n vertices are considered or the chain stops with a move into an empty cluster or a cluster consisting of only tabu vertices. So, the ejection chain idea led to a large diversification of the search while we focused on search intensification resulting from the larger neighborhood of the component moves and the chain length of n , which is the maximum possible chain length where each vertex is considered at most once. The quality of the obtained results as well as the running times let us drop the diversification part of an ejection chain based search (this, however, is completely in contrast to experiments on the traveling salesman problem).

The time complexity of the algorithm described in Figure 2 in its initialization step is $\mathcal{O}(n^2 \cdot \log n)$ and is determined by the time needed to construct n heaps. This is also the overall time complexity for a single *repeat ... until* iteration where the *for ... do* loop contributes with a complexity of $\mathcal{O}(n^2 \cdot \log n)$.

3. Computational Results of the Heuristics

We tested our algorithms on real world problem instances which are taken from Grötschel and Wakabayashi,^[24] all sources can also be found in Amorim et al.^[3] These prob-

```

Initialization:   Randomly generate a feasible solution, compute all external and
                  internal costs and construct the heaps.
repeat
  Duplicate the current solution and mark all vertices as non-tabu.
  for i := 1 to n do
    begin {Find a sequence of n best moves}
      Among all non-tabu vertices determine that one which maximizes the gain  $g(u, k)$  for all  $u$ ,
       $k \in V$ . {Intensification part}
      Let  $g(u^*, k^*)$  be maximum (not necessarily positive).
      Perform the corresponding move and mark vertex  $u^*$  tabu.
      Update all heaps and matrix  $E$ .
    end;
  {Improve the solution}
  Among all subsequences of the first  $0 \leq r \leq n$  moves of the sequence of  $n$  moves let  $r^*$  be a number
  of moves of a subsequence such that these first  $r^*$  moves improve the solution at most.
  If  $r^* > 0$  replace the current solution by its duplicate after performing these  $r^*$  moves.
until  $r^* = 0$ ;

```

Figure 2. Ejection-Chain Heuristic.

lems include the largest ones whose optimal solutions have been published in the literature. Moreover, we used 3 random data sets of 300 vertices each and a new real world problem "lecturers" of 797 vertices of which we do not know the optimal solution. The simple local search algorithm (henceforth called 1-opt) as well as the ejection chain algorithm (for short: EC) were both run 100 times each on all real world problems. Initial feasible solutions were randomly generated. All runs were performed on a Silicon Graphics Personal Iris 4D/30 computer (MIPS R3000 processor, 30 Mhz) under the operating system UNIX. The algorithms are implemented in C. In Table I we can see that all practical problems from the literature—even the larger ones—can be solved to optimality in less than 1 second on average (column " \bar{i} sec"). The third column shows the number of times the optimum was found within 100 runs.

The micro data set seems to be the most difficult one. It is the only problem instance where the optimum was not found in all runs. The 4 suboptimal results were only 0.6% below the optimum. Amorim et al.^[3] did their runs on a comparable computer, a Sun Sparc workstation, and the average computation times in both implementations, simulated annealing (parameter: number of iterations in a series is $n^2/2$, factor by which the temperature is reduced after each series of iterations equals 0.85, and the initial temperature equals 1) and tabu search (parameters: number of iterations in a series is $n/2$, and the tabu list length equals 5) were more or less the same. Their average running times for the real world problems are about 10 to 20 times higher, and there is no recognizable advantage of their simulated annealing or tabu search implementation compared to our 1-opt local search heuristic. If we sum up the running times for all real world problems, we get 2.408 seconds for the 1-opt local search heuristic, 4.157 seconds for the EC-local search heuristic, 113.16 seconds for the simulated annealing, and 89.24 seconds for the tabu search algorithm of Amorim et al.^[3]

If we consider an iteration of the EC-algorithm as one run in the *repeat...until* loop or, in other words, as a sequence of n best moves to determine the best subsequence, then each of the above problems needs between 2 and at most 5 iterations to be solved. The last iteration is

always needed to check that no further improvement of the current solution is possible. Table IIa shows the average length of the most improving subsequence (the average of the r^*) in iterations i , $i = 1, 2, 3, 4$, over 100 runs. It also shows that the maximum number of improving iteration is 2 or 3, and 4 only for the micro problem. We can conclude that many of the 100 runs already found the optimum within 1 iteration, i.e., the current solution improved only once. In particular "UNO 1b," "UNO 2b," and "companies" never used a second improving iteration for all 100 runs. Column \bar{i} of Table IIa presents the average number of iterations for each problem over 100 runs. It always includes a final non-improving iteration.

Table IIb shows the average improvement in %. The percent values are evaluated with respect to the overall improvement. Almost the whole improvement of a random initial solution is achieved during the first iteration.

We also compared the results obtained by the EC-algorithm to those that can be obtained by the 1-opt local search algorithm, which runs in $\mathcal{O}(n \cdot \log n)$ time for a single iteration. The results are shown in Table III. For each problem we used the same 100 random initial solutions as for the EC-algorithm. The third column of Table III shows how often the optimum was found within 100 runs. Column \bar{i} gives the average number of iterations needed to be locally optimal over 100 runs. In this case an iteration simply means a move. The last column \bar{t} exhibits the average time for 100 runs.

The worst solution found for the "micro" data set was 12.3% below the optimum value, while the EC-algorithm reached 0.6% below the optimum value in the worst case. The number of moves in the first iteration of the EC-algorithm empirically is of the same order as the number of moves to reach a suboptimum with respect to 1-opt local search. In both cases it is $\mathcal{O}(n)$. Hence, in both cases empirically the time complexity of both algorithms is of the same order. This also holds for the new problem instance "lecturers" to reach suboptimal solutions. Within 100 runs all suboptimal solutions from the EC-heuristic ranged from 12845 (worst solution) to 12983 (best solution found). The average of all 100 local optima is 12913, and the average running time was 45.86 seconds. The average running time over 100 runs of the 1-opt heuristic was 14.71 seconds. The results are 12951 for the best solution, 12867 on average, and 12697 for the worst solution. Both algorithms found the best solution only once. The 1-opt heuristic needed over 100 runs, on average 707.7 moves, to be suboptimal. The number of iterations of the EC-heuristic ranged from 3 to 8 (including the non-improving suboptimality check), and 4.5 on average. Table IIc describes in its first row the average number of moves of the most improving subsequences for all iterations, while the second row contains the percentage improvement on average for each iteration (rounded to two positions behind the decimal point).

Besides the real world problems we tested both heuristics on three randomly generated data sets R1, R2, and R3 (see Table IV). The entries of these data sets are randomly chosen from the sets $\{0, 1\}$, $\{0, 1, 2\}$, and $\{0, 1, 2, 3\}$, respectively, where each element of the sets has the same proba-

Table I. EC-Local Search

Problem	n	Times	\bar{i} sec
Wild cats	30	100	0.026
Cars	33	100	0.037
Workers	34	100	0.038
Cetacea	36	100	0.032
Micro	40	96	0.064
UNO	54	100	0.080
UNO 1a	158	100	0.807
UNO 1b	139	100	0.641
UNO 2a	158	100	0.930
UNO 2b	145	100	0.798
Companies	137	100	0.704

Table IIa. Average Number of Moves in Each Iteration, i.e., Average r^*

Problem	n	Iteration				\bar{i}
		1	2	3	4	
Wild cats	30	23.7	1.02	0.11	—	2.43
Cars	33	27.0	1.27	0.20	—	2.88
Workers	34	27.9	0.95	0.06	—	2.72
Cetacea	36	27.2	0.01	—	—	2.01
Micro	40	32.4	13.72	3.86	0.05	3.57
UNO	54	45.4	3.26	0.21	—	2.51
UNO 1a	158	147.9	0.09	—	—	2.06
UNO 1b	139	131.8	—	—	—	2.00
UNO 2a	158	151.1	0.11	—	—	2.07
UNO 2b	145	139.5	—	—	—	2.00
Companies	137	131.7	—	—	—	2.00

Table IIb. Average Improvement in % for Each Iteration

	Iteration			
	1	2	3	4
99.55	0.40	0.05	—	—
99.73	0.26	0.01	—	—
99.32	0.67	0.01	—	—
99.92	0.08	—	—	—
88.27	7.62	4.03	0.08	—
98.11	1.78	0.11	—	—
99.98	0.02	—	—	—
100.00	—	—	—	—
99.98	0.02	—	—	—
100.00	—	—	—	—
100.00	—	—	—	—

Table III. 1-Opt Local Search

Problem	n	Times	\bar{i}	\bar{t} sec
Wild cats	30	79	24.9	0.009
Cars	33	85	28.7	0.013
Workers	34	89	29.5	0.013
Cetacea	36	100	28.2	0.013
Micro	40	42	35.6	0.019
UNO	54	90	46.9	0.038
UNO 1a	158	100	149.0	0.481
UNO 1b	139	100	132.8	0.380
UNO 2a	158	100	152.0	0.557
UNO 2b	145	100	140.5	0.466
Companies	137	100	132.7	0.419

Table IIc. EC-Local Search on the Lecturers Problem Instance

Lecturers	Iteration						
	1	2	3	4	5	6	7
\bar{r}^*	680.4	106.87	45.25	11.93	2.18	0.75	0.04
%	97.34	2.54	0.10	0.02	0.00	0.00	0.00

Row \bar{r}^* contains the average number of moves in each iteration, i.e., average r^* .

Row % contains the average improvement in % for each iteration.

bility. To get smaller random problems too, we only took the first p rows of each of these matrices. For instance, R1. p corresponds to the problem described by the first p rows of matrix R1. Table IV exhibits the results obtained by both algorithms for 15 random problems of size 25 up to 300 objects (rows). The "best" column contains the best result, and enclosed in brackets is the number of times this result is obtained within 100 runs. The \bar{x} column gives the aver-

age of all 100 local optima, and column "worst" presents the worst results. Both algorithms got the same set of 100 randomly generated initial solutions for each of the 15 problems.

The results confirm the superiority of the EC-heuristic to the 1-opt local search heuristic. Although the average running time over 100 runs \bar{t} for the EC-heuristic is between 3 and 5 times the average running time of 1-opt local search, the empirical times complexity of both algorithms is about $n^2 \cdot \log n$.

Compared to the real world problems the random problems of Table IV seem to be more difficult. The reason may be that the random problems are completely unstructured. The edge lengths connecting the vertices of the random problems are almost uniformly distributed, hence almost equal. There are no clusters inherited when the object-attribute structure is translated into a weighted graph. Thus, it is not a surprise that the algorithms presented above behave completely differently because they try to find clusters where in fact there are none in the underlying data. Tests based on these type of random problems are less useful to draw conclusions about the behavior of heuristics on practical data. They demonstrate, however, the robust-

Table IV. Results of the Random Problems (100 Runs)

Problem	EC-Heuristic				1-opt-Heuristic			
	Best (Times)	\bar{x}	Worst	\bar{t} sec	Best (Times)	\bar{x}	Worst	\bar{t} sec
R1.25	255(45)	252.9	251	0.02	255(43)	250.7	185	0.006
R1.50	834(12)	822.1	788	0.09	834(1)	803.7	710	0.02
R1.100	3173(1)	3089.5	2949	0.57	3124(1)	3022.2	2873	0.13
R1.200	11525(6)	11177.1	10259	2.85	11455(3)	10846.3	9750	0.62
R1.300	24896(27)	24503.0	23035	6.92	24896(3)	23861.1	22556	1.52
R2.25	39(60)	38.6	38	0.02	39(28)	38.0	35	0.005
R2.50	147(79)	146.8	145	0.08	147(60)	145.8	138	0.02
R2.100	326(6)	321.9	313	0.46	323(3)	315.5	296	0.14
R2.200	1030(1)	1009.9	976	2.58	1020(1)	967.5	923	0.73
R2.300	1959(1)	1901.6	1848(1)	7.24	1878(1)	1804.8	1735	1.88
R3.25	5(100)	—	—	0.02	5(100)	—	—	0.007
R3.50	17(100)	—	—	0.10	17(68)	16.5	14	0.03
R3.100	76(36)	75.4	75	0.61	75(15)	73.1	68	0.21
R3.200	252(22)	250.0	243	3.00	252(1)	243.4	243	1.03
R3.300	487(3)	480.3	470	7.49	480(2)	467.0	442	2.55

ness of both heuristics. The EC-heuristic seems to be more robust with respect to changes of data than the 1-opt local search heuristic.

4. An EC-Based Branch and Bound Algorithm

The success of the heuristics has motivated the development of a branch and bound algorithm for the clique-partitioning problem. We first outline the algorithm and then fill in the details in the subsequent sections.

The formulation of the clique-partitioning problem in the second chapter suggests a binary structure of a branch and bound tree. At the root all x_{ij} are still free; at each branch a free edge (i, j) is either *included* (fix x_{ij} to 1) or *excluded* (fix x_{ij} to 0).

Our method for systematically exploring the tree is depth-first search, or backtracking. Although it is not immediately apparent from the structure of the search tree, the algorithm distinguishes between two types of branches:

Explicit branches correspond to “free” decisions made according to a rule described below.

Implicit branches are a logical consequence of explicitly or implicitly fixing other edges. Suppose, for example, that after including an edge (i, j) we now have $x_{ij} = 1$ and $x_{ik} = 0$, while x_{jk} is still free. Since a solution must be transitive x_{jk} has to be 0.

Instead of explicit or implicit branches we will occasionally speak of explicit or implicit edges or variables, meaning the edge (variable) fixed at the branch.

Figure 3 shows the pseudo code for the outline of the search. The notation `functionName(&argument)` indicates a “call by reference”: The argument may be (here it will be) changed in the subroutine.

At the beginning, the function `testsBeforeBranching` calculates a lower and an initial upper bound for the value of

```

testsBeforeBranching();
chooseAndFixFirstEdge(&newFixedEdge);
done := FALSE;
while not done do
  if (logicalTests(newFixedEdge)) then
    branch(&newFixedEdge);
  else
    done := backtrack(&newFixedEdge);

```

Figure 3. Outline of the EC-based branch and bound search.

the optimal solution. It then tries to fix some edges implicitly by employing a logical test before explicit branching begins. The function `chooseAndFixFirstEdge` does what its name indicates: It selects and fixes an edge for the first explicit branch. Afterwards, the depth-first search loop is entered.

The routine `logicalTests` performs two tasks: It implicitly fixes edges as a consequence of the last explicit branching decision and it tests if the current node of the search tree is fathomed by comparing the new upper bounds—we use two independent bounds—with the best lower bound. If the node cannot be fathomed `logicalTests` returns TRUE and the search proceeds with branch where a new edge is explicitly fixed. If a node is fathomed or if the search arrives at a leaf of the tree, `logicalTests` returns the value FALSE and backtracking begins. The function `backtrack` goes back through the tree until it arrives at the most recently fixed explicit edge. All implicit variables encountered along the way are set free again. The explicit edge is then fixed to its opposite value and marked as *implicit*. It will be set free the next time it is visited again during backtracking. The result of `backtrack` is always FALSE, except when we return to the root of the tree and nothing is left to explore (i.e., the first branch is marked as *implicit*); then `done` is set to TRUE and the search stops. The best solution that has been found is

optimal. Next, we take a closer look at the individual components of the algorithm.

A Simple Upper Bound

We want to maximize the sum of the edge weights within all cliques. An upper bound \bar{z} for this sum follows from the relaxation of our model in the second section, obtained by disregarding the triangle inequalities. The initial bound $\bar{z}^{(0)}$ at the root of the search tree (node 0) is simply the sum of all positive edge weights. Whenever an edge with positive weight—a positive edge for short—is excluded or a negative edge is included this upper bound decreases. At a node λ , somewhere in the tree, we have

$$\bar{z}^{(\lambda)} = \bar{z}^{(0)} - \sum_{\substack{1 \leq i < j \leq n \\ x_{ij} \text{ fixed}}} ((1 - x_{ij}) \cdot \max\{w_{ij}, 0\} - x_{ij} \cdot \min\{w_{ij}, 0\}).$$

This is the upper bound used in Tüshaus^[53] and Amorim et al.^[3] Our computational experience has shown that this bound alone cannot efficiently limit the search space. After describing an initial logical test based on the triangle inequalities we will develop another, triangle-based bound.

An Initial Logical Test

At the outset of the algorithm, the objective function value obtained by the EC-heuristic serves as a lower bound \underline{z} ; an upper bound $\bar{z}^{(0)}$ is determined as in the preceding section. The tests described here are then applied before branching. They try to show that certain edges must be included or excluded in a solution with a value of at least \underline{z} . The tests use the triangle inequalities of the model which ensure that a solution is transitive: For all triangles $\nabla(i, j, k)$ they prohibit every combination of the corresponding binary variables in which exactly 2 variables have the value 1. If an edge (i, j) is included ($x_{ij} = 1$), we can draw a conclusion about every triangle $\nabla(i, j, k)$ in which the edge appears:

$$x_{ij} = 1 \text{ yields } \begin{cases} x_{ik} = 1 \text{ and } x_{jk} = 1, \\ \text{or} \\ x_{ik} = 0 \text{ and } x_{jk} = 0. \end{cases} \text{ for all } k \in V \setminus \{i, j\}.$$

Similarly, if (i, j) is excluded then

$$x_{ij} = 0 \text{ yields } \begin{cases} x_{ik} = 0 \text{ and } x_{jk} = 0, \text{ or} \\ x_{ik} = 1 \text{ and } x_{jk} = 0, \text{ or} \\ x_{ik} = 0 \text{ and } x_{jk} = 1. \end{cases} \text{ for all } k \in V \setminus \{i, j\}.$$

These observations can be used to design logical tests. We only describe the test for $x_{ij} = 1$; the test for the opposite assumptions $x_{ij} = 0$ works similarly.

The question to be answered is: What is the value of the best solution—or: find a bound for this value—that can be achieved if x_{ij} is set to 1? If the value is less than \underline{z} , then the edge (i, j) cannot be included in an optimal solution

and we may directly fix x_{ij} to 0. In other words, the test of $x_{ij} = 1$ tries to show that $x_{ij} \neq 1$ in an optimal solution.

Consider a triangle $\nabla(i, j, k)$: If $x_{ij} = 1$ we have the choice either to exclude both edges (i, k) and (j, k) or to include them. If we exclude a positive or include a negative edge the initial upper bound $\bar{z}^{(0)}$ decreases. We call this decrease $\delta_{ijk}(x_{ij}; x_{ik}, x_{jk})$. For $x_{ij} = 1$ we obtain:

$$\delta_{ijk}(1; 0, 0) = \max\{w_{ik}, 0\} + \max\{w_{jk}, 0\}, \text{ and}$$

$$\delta_{ijk}(1; 1, 1) = -\min\{w_{ik}, 0\} - \min\{w_{jk}, 0\}.$$

An edge (i, j) appears in $n - 2$ triangles. If, after setting $x_{ij} = 1$, we make the best choice for the remaining 2 edges in each of these triangles; then the sum of the $n - 2$ corresponding δ values is the amount $\Delta_{ij}(1) := \Delta_{ij}(x_{ij} = 1)$ by which the upper bound $\bar{z}^{(0)}$ will at least decrease:

$$\Delta_{ij}(1) = \sum_{k \in V \setminus \{i, j\}} \min\{\delta_{ijk}(1; 0, 0), \delta_{ijk}(1; 1, 1)\}.$$

If the upper bound decreases below the lower bound \underline{z} , then the edge (i, j) cannot be included in an optimal solution, hence:

$$\bar{z}^{(0)} + \min\{w_{ij}, 0\} - \Delta_{ij}(1) < \underline{z} \text{ yields } x_{ij} \neq 1.$$

The term $\min\{w_{ij}, 0\}$ takes into account the decrease in the initial bound caused by including a negative edge. It is easy to devise a similar test that uses analogous values $\Delta_{ij}(0)$ and that tries to show that an edge must be included in an optimal solution.

What is the complexity of this test? Since an edge (i, j) appears in $\mathcal{O}(n)$ triangles, and a single δ value can be found in constant time, the overall effort for finding Δ_{ij} is $\mathcal{O}(n)$. Given \underline{z} , testing all $\binom{n}{2}$ edges requires effort $\mathcal{O}(n^3)$.

A Second Upper Bound

The algorithm uses a second, triangle-based upper bound \hat{z} besides \bar{z} . At the root $\hat{z}^{(0)} = \bar{z}^{(0)}$ but at an arbitrary node the bounds will usually differ.

The idea behind \hat{z} is the same as the one used in the initial tests. After explicitly or implicitly fixing a variable x_{ij} we try to answer the question: What is the best choice for the remaining 2 edges—provided they are still free—in all triangles in which the edge (i, j) appears? Again, we only describe the argument for the case that x_{ij} is set to 1.

Using the δ values from the preceding section we can define an estimate (lower bound) $\Delta_{ij}^*(1) := \Delta_{ij}^*(x_{ij} = 1)$ for the decrease of \hat{z} caused by setting $x_{ij} = 1$:

$$\Delta_{ij}^*(1) = \sum_{\substack{k \in V \setminus \{i, j\} \\ (i, k), (j, k) \text{ free} \\ \text{and not marked}}} \min\{\delta_{ijk}(1; 0, 0), \delta_{ijk}(1; 1, 1)\}.$$

The definition is almost the same as for Δ_{ij} , the difference being that only those triangles are used in which edges (i, k) and (j, k) are both free and not marked. If $\min\{\delta_{ijk}(1; 0, 0), \delta_{ijk}(1; 1, 1)\} > 0$ then the triangle $\nabla(i, j, k)$ can be used to lower the bound \hat{z} ; in this case the edges (i, j) and (j, k) are marked and stored in a list kept at the current node of the tree. The marked-flags are needed to

prevent multiple use of the same edge. During backtracking the marks stored at a node are removed again.

Let μ be the predecessor of a node λ in the search tree; if λ is reached by (explicitly or implicitly) including (i, j) then:

$$\hat{z}^{(\lambda)} = \hat{z}^{(\mu)} + \min\{w_{ij}, 0\} - \Delta_{ij}^*(1)$$

if edge (i, j) is not marked.

A similar argument can be used to find \hat{z} for a node reached by exclusion of an edge. Note that the bound \hat{z} is independent of \bar{z} ; the tighter one may serve to fathom a branch.

Explicit Branching

Only positive edges are used for explicit branching. Once all positive edges are fixed, the remaining free variables can be set to 0; as we always make all necessary implicit branches, this will not violate the transitivity constraints. To select a free edge for branching we have—after some tests—chosen the following heuristic rule: Fix the free positive edge for which the expression

$$\Delta_{ij}(\bar{x}_{ij}^{(h)}) + \max\{|V_{v(i)}|, |V_{v(j)}|\}$$

is maximized; $x_{ij}^{(h)}$ is the binary status of edge (i, j) in the EC-heuristic solution and $\bar{x}_{ij}^{(h)}$ is the negation of this value; $v(i)$ is the number of the clique containing vertex i in this solution, and $|V_{v(i)}|$ is the vertex cardinality of this clique. The Δ_{ij} are the same as in the initial logical tests. The rationale of this rule is: The first term favors edges for which the upper bound \hat{z} is likely to decrease significantly when the search proceeds in the presumably wrong direction. The second term favors edges with an endpoint in a large cluster; on average this leads to a greater number of implicit branches.

Implicit Branching

After explicitly fixing an edge (i, j) all triangles $\nabla(i, j, k)$, $k \in V \setminus \{i, j\}$, are inspected to find the necessary implicit branches. Because an implicit branch may cause further implicit branching, every new implicit edge is added to a list. When all triangles containing (i, j) have been checked the same process is repeated for all edges in the list.

The deeper the search has proceeded into the tree, the higher the number of implicit branches resulting from a single explicit branching decision will be on average. This is because a new implicit branch can only be made in a triangle where 2 edges are already fixed.

5. Computational Results of the EC-Based Branch and Bound Algorithm

As the local search heuristics, the branch and bound algorithm has been implemented in C on a Silicon Graphics Personal Iris 4/D 30 (MIPS R3000 processor, 30 Mhz) workstation under Unix. It has been tested using the 11 problems from Table I.

Table V shows the initial upper bound $\bar{z}^{(0)}$ the value of the optimal solution z^* and the solution time including

Table V. Results of the EC-based Branch and Bound Algorithm

Problem	n	$\bar{z}^{(0)}$	z^*	$t \text{ sec}^a$	$t \text{ sec}^b$
Wild cats	30	1400	1304	0.5	23
Cars	33	1748	1501	5.3	155
Workers	34	1233	964	29.8	198
Cetacea	36	998	967	0.2	15
Micro	40	1362	1034	898.1	257
UNO	54	906	778	128.1	270
UNO 1a	158	12322	12197	22.3	852
UNO 1b	139	11859	11775	11.8	518
UNO 2a	158	73178	72820	19.1	587
UNO 2b	145	72111	71818	13.5	484
Companies	137	82691	81874	11.3	1187

^aComputer: Silicon Graphics Personal Iris 4D/30, under Unix.

^bComputer: Siemens 7.865, under VM/370-CMS; with MPSX/370 (from Grötschel and Wakabayashi^[24]).

the time required for the initial EC-heuristic solution. For comparison the last column cites the time required by the branch and cut method described in Grötschel and Wakabayashi.^[24,25] A direct comparison of the running times obviously suffers from the different hardware performance characteristics, but even then it seems safe to conclude that our method is often significantly faster.

The computational results also show that the time required by our algorithm can vary considerably. It comes as no surprise that the problems for which the ratio $\bar{z}^{(0)}/z^*$ is relatively large (cars, workers, micro, UNO) are the most difficult ones. All running times on the real world problems (except the micro instance) for our branch and bound implementation sum up to 241.9 seconds, which is only about twice the time needed in the simulated annealing or tabu search approach described in Amorim et al.^[3]

Although we got the same clusters in all optimal solutions of the real world problems, our objective function values in the optimal solution differ in four cases (micro, UNO, UNO 1b, and companies) from those obtained by Grötschel and Wakabayashi^[24] or Amorim et al.^[3] These differences already might be caused by a typo in the data matrix. Our objective function values for UNO, UNO 1b, and companies are 778, 11775, and 81874, respectively, while Grötschel and Wakabayashi got 798, 11613, and 81802, respectively. In our case in the micro instance the right-hand side for the edge weight computations has to be 0.262—in order to reach the same objective function value.

We were not able to solve the lecturers problem instance to optimality within a time limit of 2000 seconds. Thus, this problem does not appear in Table V.

6. Conclusion

We have presented a powerful local search heuristic and effective branch and bound algorithm for the clique-partitioning problem. Both algorithms did their job very well in

case of the real world problems known from literature. Randomly generated problems often can tell us about the robustness of a solution method; however, as long as there is no real world structure underlying the random data we cannot draw conclusions about the usefulness of these methods. Test beds of real world problems are in general indispensable.

A branch and cut algorithm need not be the most powerful tool to solve problems exactly. In our case it was dominated by the branch and bound procedure employing a simple propagation of constraints in order to reach arc consistency (cf. Han and Lee^[27]).

The algorithms of Kernighan and Lin^[35,39] can be seen as an application of tabu search with dynamically organized and increasing tabu list. The generalization in form of the ejection chain method (Glover^[22,23]) provides an extremely powerful tool for solving combinatorial optimization problems. The ejection chain idea can easily be applied without substantial changes to other partitioning problems mentioned in the introduction, as well as to related facility location problems. In our case even the rudimentary implementation of an ejection chain solved all real world problems. The complete version could not do faster, however, for large problem instances it may be indispensable.

Appendix: Classification of Lecturers

The Association of Business Administration Lecturers consists of 13 commissions (see below). Each "1" entry of the data matrix expresses either membership of a commission or interest in its subject. The last attribute (no. 27) represents interest in logistics of which there is no commission. In the listing of the data below for each object (i.e., lecturer) we only give the numbers of those columns corresponding to attributes with a "1" entry in the 0,1 data matrix. The number of the object is always printed in bold face.

Specification of All 27 Attributes

column	attribute
1/2	Member of the commission/Interest in International Management
3/4	Member of the commission/Interest in Information Management
5/6	Member of the commission/Interest in Operations Research
7/8	Member of the commission/Interest in Production Management
10/9	Member of the commission/Interest in Marketing
11/12	Member of the commission/Interest in Science Theory
13/14	Member of the commission/Interest in Banking and Finance
15/16	Member of the commission/Interest in Controlling
17/18	Member of the commission/Interest in Accountancy
19/20	Member of the commission/Interest in Personnel Management
21/22	Member of the commission/Interest in Public Administration
23/24	Member of the commission/Interest in Environmental Economics
25/26	Member of the commission/Interest in Organization Theory
27	Interest in Logistics

Data

1. 1 2 9 10; 2. 3 4 6; 3. 5 6 8 16 18; 4. 1 2 14 15 16; 5. 6 26 27; 6. 7 8 19 20 26; 7. 3 4 16 26; 8. 16; 9. 5 6; 10. 6 8 9 14; 11. 3 4 15 16 17 18; 12. 3 4 5 6 7 8 27; 13. 4 8; 14. 2 9 10; 15. 4 6 8 12 16; 16. 5 6 7 8; 17. 3 4 9 10; 18. 19 20 26; 19. 4 5 6 7 8 26; 20. 23 24 26; 21. 6 8 14 27; 22. 15 16; 23. 2 14 16; 24. 19 20 25 26; 25. 9 11 12 14 22 25 26; 26. 6 13 14 15 16 17 18; 27. 4 6 20 25 26; 28. 19 20 24 25 26; 29. 3 4 6 8 27; 30. 7 8 25 26; 31. 15 16 18; 32. 8 27; 33. 16 17 18; 34. 14 16 17 18; 35. 4 8; 36. 5 6 7 8 16; 37. 6 14 25 26; 38. 7 8 25 26; 39. 1 2 26; 40. 9 25 26; 41. 22; 42. 9 10; 43. 4; 44. 6 13 14; 45. 6 7 8 15 16 27; 46. 3 4 5 6 9 10; 47. 9; 48. 9 18; 49. 1 2 25 26; 50. 6 8 16; 51. 9; 52. 9 10 26; 53. 20; 54. 9 10 12; 55. 15 16; 56. 8 27; 57. 3 4 26; 58. 16 18; 59. 2 9 10 24; 60. 3 4; 61. 1 2 19 20 25 26; 62. 6 13 14; 63. 3 4 5 6 16; 64. 15 16 22 27; 65. 15 16 25 26; 66. 6; 67. 5 6 7 8 27; 68. 4 16 26; 69. 22; 70. 18; 71. 8; 72. 3 4 19 20 25 26; 73. 7 8 9 10 15 16 23 24; 74. 14 15 16 17 18; 75. 1 2 11 12 19 20 24 25 26; 76. 15 16; 77. 1 2 17 18; 78. 11 12 19 20 25 26; 79. 13 14 15 16; 80. 9 10; 81. 6 8; 82. 14 16; 83. 15 16 17 18; 84. 4 6 26; 85. 16 18; 86. 6 16; 87. 14 16 18; 88. 26; 89. 3 4 5 6 7 8 27; 90. 20 25 26; 91. 6 9; 92. 11 12 26; 93. 3 4 25 26; 94. 13 14; 95. 15 16 18; 96. 4 6 16 26; 97. 4 9 10; 98. 3 4 6 7 8 14 15 16; 99. 9 10; 100. 9 10; 101. 5 6 13 14; 102. 19 20 25 26; 103. 1 2 6 9 10 14; 104. 19 20 26; 105. 21 22; 106. 1 2 9 10; 107. 8; 108. 16 26; 109. 3 4 6 26; 110. 19 20 21 22 25 26; 111. 21 22; 112. 9 10; 113. 9 10; 114. 16 17 18 20 25 26; 115. 20; 116. 5 6 7 8 9 10 15 16 27; 117. 2 14 26; 118. 2; 119. 16; 120. 26; 121. 3 4 5 6 12 26; 122. 19 20 25 26; 123. 8 13 14; 124. 9 10; 125. 4; 126. 13 14; 127. 12 16 18; 128. 6 8 12 14 15 16 18 27; 129. 14 18; 130. 8 15 16 21 22; 131. 15 16 21 22; 132. 19 20 25 26; 133. 7 8 21 22; 134. 3 4 5 6 7 8; 135. 19 20 24 26; 136. 5 6 7 8 23 24; 137. 8 16; 138. 1 2 6 11 12 21 22 25 26; 139. 19 20 25 26; 140. 6 14; 141. 13 14 15 16 21 22; 142. 13 14 16; 143. 13 14; 144. 21 22 25 26; 145. 1 2 13 14; 146. 3 4; 147. 6 9 10; 148. 6 7 8; 149. 5 6; 150. 9 10 21 22; 151. 7 8 19 20 25 26; 152. 8 12 16 26; 153. 9 10 14 15 16; 154. 6 8 14

27; 155. 3 4 6 26; 156. 2 19 20 25 26; 157. 6 8 27; 158. 4 14; 159. 9 10 12 14; 160. 15 16 18 20; 161. 7 8; 162. 20 25 26; 163. 1 2 4 11 12; 164. 3 4; 165. 9 10 26; 166. 1 2 9 10; 167. 16 17 18; 168. 15 16; 169. 6 8 14 15 16 17 18; 170. 1 2 8 11 12 25 26; 171. 6 15 16 21 22; 172. 14 15 16; 173. 16 17 18; 174. 20 23 24 25 26; 175. 14 15 16 17 18; 176. 3 4; 177. 6 7 8 27; 178. 6 8; 179. 7 8 15 16 22 26; 180. 1 2 9 10; 181. 3 4 5 6 26; 182. 20 22 26; 183. 16; 184. 2 9 14; 185. 14; 186. 7 8 14; 187. 3 4; 188. 3 4 16 25 26; 189. 19 20; 190. 17 18; 191. 4 25 26; 192. 1 2 9 10 26; 193. 16 26; 194. 16; 195. 11 12 20 25 26; 196. 16 17 18; 197. 13 14; 198. 3 4 8 26; 199. 9 10; 200. 9 10 26; 201. 4 26; 202. 3 4 20 25 26; 203. 1 2 15 16; 204. 3 4 6; 205. 12 20 25 26; 206. 7 8 27; 207. 1 2 9 10 11 12; 208. 11 12 13 14 15 16 17 18; 209. 5 6 8 16; 210. 14 17 18; 211. 9; 212. 1 2 4 13 14; 213. 11 12 13 14; 214. 13 14; 215. 9 10 20; 216. 1 2 9 10; 217. 6 14 16; 218. 6 15 16; 219. 13 14 15 16; 220. 3 4 5 6; 221. 9; 222. 1 2 9; 223. 15 16 17 18; 224. 18; 225. 16 21 22; 226. 13 14 19 20; 227. 6 14 17 18; 228. 3 4 6 7 8; 229. 4 6 21 22; 230. 11 12 19 20 25 26; 231. 11 12 19 20 25 26; 232. 19 20 25 26; 233. 4 8 16; 234. 2 19 20 21 22 25 26; 235. 9 10 14; 236. 8 25 26; 237. 4 6; 238. 7 8 20 26; 239. 13 14 15 16; 240. 13 14 15 16 21 22; 241. 9 10 25 26; 242. 3 4 6; 243. 17 18; 244. 5 6 7 8 14 16; 245. 14; 246. 8; 247. 4; 248. 9 10; 249. 16 20 26; 250. 2 26; 251. 9 22 26; 252. 6 7 8; 253. 21 22 26; 254. 7 8 9 15 16; 255. 19 20 25 26; 256. 5 6; 257. 19 20 21 22 23 24 25 26; 258. 9 10 26; 259. 11 12 19 20 25 26; 260. 3 4 21 22; 261. 1 2 19 20 25 26; 262. 3 4 7 8 16; 263. 8 15 16 27; 264. 2 9; 265. 2 13 14 15 16 21 22; 266. 6 7 8 16 26 27; 267. 11 12 19 20 21 22 25 26; 268. 4; 269. 9 10 11 12; 270. 26; 271. 19 20 25 26; 272. 21 22 24 26; 273. 26; 274. 13 14; 275. 3 4 6 8 16 26; 276. 14 16 18 26; 277. 19 20 21 22 25 26; 278. 5 6; 279. 4 8; 280. 9 17 18; 281. 16 26; 282. 14 16 26; 283. 9 10 15 16; 284. 9; 285. 16 18 26; 286. 3 4 25 26; 287. 1 2 20 25 26; 288. 3 4 7 8 11 12 25 26; 289. 1 2 6 25 26 27; 290. 7 8 26; 291. 3 4 5 6 7 8; 292. 16; 293. 1 4 14; 294. 1 2 14 15 16; 295. 15 16 23 24; 296. 14 26; 297. 1 2 14; 298. 16 17 18; 299. 3 4 6 27; 300. 5 6; 301. 1 2 19 20 25 26; 302. 3 4; 303. 5 6 15 16 21 22; 304. 15 16 22 25 26; 305. 6 9 10; 306. 21 22; 307. 19 20 21 22 25 26; 308. 6 27; 309. 14 16; 310. 14 16; 311. 20 26; 312. 3 4 8 26; 313. 19 20 25 26; 314. 19 20 26; 315. 13 14; 316. 2 16; 317. 4 6 9 10 16; 318. 3 4; 319. 4 16 26; 320. 9 10; 321. 20; 322. 14 16; 323. 14 16 18 26; 324. 25 26; 325. 3 4 5 6; 326. 4 26; 327. 1 2 9 10 15 16; 328. 9 10; 329. 4 25 26; 330. 19 20 25 26; 331. 6 9 10 25 26; 332. 1 2 9; 333. 9; 334. 3 4; 335. 14 21 22; 336. 1 2 9 10; 337. 16 26; 338. 8 14; 339. 15 16 18; 340. 1 2 14; 341. 14; 342. 3 4 16; 343. 4 13 14 15 16 18; 344. 3 4; 345. 4 7 8 15 16; 346. 4 8 14 16; 347. 14 16 17 18; 348. 9; 349. 4 5 6 13 14 27; 350. 9 10; 351. 5 6 21 22; 352. 14 16; 353. 9 10 23 24; 354. 3 4; 355. 1 2 14 26; 356. 13 14 15 16; 357. 16 17 18; 358. 1 2 9 10; 359. 9 10 26; 360. 6 9 10 25 26; 361. 9 10; 362. 19 20 25 26; 363. 14; 364. 7 8 16; 365. 16; 366. 15 16 21 22; 367. 4 6 8; 368. 9 10 14 16; 369. 19 20 26; 370. 1 2 3 4 9 10; 371. 6 9 16 19 20 25 26; 372. 7 8 15 16 21 22 27; 373. 17 18; 374. 16; 375. 16; 376. 26; 377. 9 26; 378. 12 14 16; 379. 9 10 19 20; 380. 5 6 19 20 25 26; 381. 1 2 16 19 20 25 26; 382. 16; 383. 15 16 18; 384. 14 15 16 21 22; 385. 7 8 15 16 26; 386. 1 2 15 16; 387. 17 18; 388. 13 14; 389. 6 14; 390. 3 4 5 6 13 14; 391. 1 2 13 14; 392. 1 2 8 15 16 25 26 27; 393. 9 10 20 26; 394. 4 26; 395. 6 9 10 27; 396. 4 5 6 7 8; 397. 4 16 17 18; 398. 4 9 10; 399. 1 2 9 10; 400. 5 6 7 8 14; 401. 3 4 26; 402. 15 16 17 18 21 22; 403. 25 26; 404. 2 16; 405. 6 7 8 15 16; 406. 6 7 8 14 15 16; 407. 6 14 25 26; 408. 19 20 25 26; 409. 13 14 15 16; 410. 13 14 15 16 17 18; 411. 14 17 18; 412. 9 10; 413. 1 2 9 10 26; 414. 16; 415. 6 14; 416. 1 2 25 26; 417. 26; 418. 3 4; 419. 15 16 17 18; 420. 6 21 22 25 26; 421. 7 8 15 16 26; 422. 9 10 14; 423. 1 2 20 26; 424. 1 2 9 10; 425. 16; 426. 4 9 10; 427. 6 25 26; 428. 9 10; 429. 4 11 12 19 20 25 26; 430. 6 13 14 15 16; 431. 13 14 16; 432. 2 9 10 19 20 27; 433. 25 26; 434. 18 26; 435. 16 26; 436. 13 14 16; 437. 9 10 12; 438. 19 20 26; 439. 1 2 7 8 25 26; 440. 3 4 7 8; 441. 16 17 18; 442. 4 21 22 26; 443. 14; 444. 3 4 6; 445. 11 12 13 14 26; 446. 14 16; 447. 19 20 25 26; 448. 5 6 7 8; 449. 1 2 7 8 9 10 26; 450. 3 4 5 6; 451. 3 4 8 16; 452. 6 13 14 17 18; 453. 3 4 7 8 16 26; 454. 13 14 16; 455. 9 10 25 26; 456. 14 16; 457. 15 16; 458. 14; 459. 2 26; 460. 14 15 16 18; 461. 3 4 7 8 27; 462. 1 2 9 10; 463. 6 8 15 16; 464. 19 20 21 22 25 26; 465. 12 17 18; 466. 9 10; 467. 9 10; 468. 16; 469. 16; 470. 5 6 7 8; 471. 6 11 12 16 26; 472. 9 10; 473. 13 14 16; 474. 19 20 25 26; 475. 5 6 7 8 26; 476. 3 4; 477. 20; 478. 6; 479. 3 4; 480. 11 12 19 20 25 26; 481. 1 2 19 20 25 26; 482. 7 8 14 15 16 19 20 21 22; 483. 9; 484. 14 16 17 18; 485. 6 8 11 12 25 26; 486. 16; 487. 3 4 9 10; 488. 4 6; 489. 16; 490. 1 2 13 14 15 16; 491. 9 10; 492. 7 8 15 16; 493. 3 4 6 26; 494. 3 4 6 14 25 26; 495. 3 4 6 8; 496. 4 14 16 17 18; 497. 6 7 8 18 26; 498. 5 6 27; 499. 1 2 5 6 8 14 27; 500. 6 27; 501. 4 6 9 10; 502. 9 10 12 16 26; 503. 3 4 5 6; 504. 9 15 16 17 18; 505. 3 4 14 15 16; 506. 7 8; 507. 3 4 5 6 9 10 27; 508. 26; 509. 1 2 4 15 16; 510. 9 10 26; 511. 9; 512. 14 26; 513. 15 16; 514. 1 2 9 10; 515. 1 2 15 16; 516. 14 16; 517. 7 8 15 16; 518. 4 14 16 26; 519. 1 2 4 19 20; 520. 16 25 26; 521. 4 13 14; 522. 7 8 26; 523. 9 10 14 25 26; 524. 8 9 10 14 16; 525. 6 9 10 14; 526. 3 4; 527. 13 14; 528. 15 16 17 18; 529. 9 10; 530. 8 16; 531. 8 19 20 21 22 25 26; 532. 6; 533. 6 14; 534. 16 20; 535. 3 4 25 26; 536. 3 4 26; 537. 14 15 16 17 18; 538. 16; 539. 14 16 17 18 24; 540. 1 2 3 4 5 6 21 22; 541. 13 14; 542. 14 16 25 26; 543. 7 8; 544. 5 6; 545. 9 10 22; 546. 3 4 26; 547. 6 8 9 10; 548. 6 13 14 15 16; 549. 5 6; 550. 3 4 5 6; 551. 4 6 7 8 9 10; 552. 9 10; 553. 5 6 16 21 22; 554. 3 4; 555. 8 26 27; 556. 6 8 14; 557. 5 6 9 10 14; 558. 6 19 20 25 26; 559. 6; 560. 16; 561. 9 14 26; 562. 14; 563. 7 8 15 16 26; 564. 14; 565. 2 14 16 18 22; 566. 14 17 18; 567. 14 15 16 17 18; 568. 4 26; 569. 4 5 6 7 8; 570. 15 16 17 18; 571. 16 26; 572. 3 4 6 7 8; 573. 9 10 15 16; 574. 9 10; 575. 8 25 26; 576. 4 9 10; 577. 3 4 6 16; 578. 3 4 25 26; 579. 3 4; 580. 1 2 13 14; 581. 15 16 27; 582. 25 26; 583. 7 8 14 15 16; 584. 9 10; 585. 9 10; 586. 13 14; 587. 9 10 19 20 25 26; 588. 9 10; 589. 1 2 13 14; 590. 4 6; 591. 1 2 13 14 15 16; 592. 4 5 6 9 10; 593. 12 19 20 25 26; 594. 19 20 25 26; 595. 19 20 26; 596. 19 20 25 26; 597. 16; 598. 3 4 5 6; 599. 7 8 16; 600. 4 19 20 25 26; 601. 14 18; 602. 14 16 24; 603. 4 14 20 22; 604. 7 8 15 16; 605. 14 16 18; 606. 19 20 25 26; 607. 9 10; 608. 8 19 20 25 26; 609. 11 12 23 24; 610. 16; 611. 15 16 17 18; 612. 16 17 18; 613. 14 15 16 18; 614. 6 13 14; 615. 12 16; 616. 14 15 16; 617. 7 8 16; 618. 9 10 11 12 15 16; 619. 14 16; 620. 3 4 5 6; 621. 13 14; 622. 1 2 14 17 18; 623. 3 4 16; 624. 8 14 16 22 26; 625. 4 16; 626. 14; 627. 6 7 8 14 26; 628. 14; 629. 6 13 14 15 16; 630. 15 16 18; 631. 25 26; 632. 8 16 26; 633. 9; 634. 13 14; 635. 13 14 26; 636. 9 10 14; 637. 1 2; 638. 8 14 18; 639. 6 7 8 16; 640. 14 16 17 18; 641. 14 16 17 18; 642. 11 12 16 21 22; 643. 14 15 16; 644. 21 22; 645. 21 22 24; 646. 3 4; 647. 15 16; 648. 16 17 18; 649. 9 19 20 26; 650. 14 16 26; 651. 3 4 19 20 25 26; 652. 7 8 23 24; 653. 23 24 26; 654. 1 2 11 12 25 26; 655. 1 2 19 20 25 26; 656. 6 19 20 25 26; 657. 13 14 15 16 25 26; 658. 3 4 5 6 7 8 27; 659. 2 11 12 26; 660. 8 16 26; 661. 15 16 17 18; 662. 5 6 7 8 27; 663. 19 20 25 26; 664. 12 25 26; 665. 16 18; 666. 6 8; 667. 4 5 6 27; 668. 6 9 10; 669. 16 18 21 22; 670. 3 4 6; 671. 6 9 10 26; 672. 2 13 14; 673. 17 18; 674. 15 16; 675. 6 7 8 9 10 27; 676. 6 7 8 14 16; 677. 20 26; 678. 9 26; 679. 3 4 6 8 16; 680. 18; 681. 19 20 25 26; 682. 7 8; 683. 8 16; 684. 19 20 25 26; 685. 6 15 16 26; 686. 19 20; 687. 13 14; 688. 12 14 15 16 22 25 26; 689. 26;

690. 4 5 6; 691. 15 16 18; 692. 6 8 16; 693. 1 2 14 15 16 25 26; 694. 13 14; 695. 16; 696. 8 16; 697. 19 20 25 26; 698. 5 6 13 14; 699. 9 13 14; 700. 4 6 14 27; 701. 11 12 16 21 22 25 26; 702. 14 16 18; 703. 6 16 21 22; 704. 9 10; 705. 19 20 25 26; 706. 3 4 5 6 7 8 15 16; 707. 4 6 8 16; 708. 6 11 12 14 15 16; 709. 1 2 14 19 20 25 26; 710. 19 20 25 26; 711. 21 22; 712. 2 11 12 25 26; 713. 8 20; 714. 11 12 21 22 25 26; 715. 27; 716. 1 2 24 25 26; 717. 7 8 9 10 13 14 15 16; 718. 16 21 22; 719. 4 8 16 26; 720. 16 17 18; 721. 5 6 7 8; 722. 20 26; 723. 9 10; 724. 3 4; 725. 15 16 18; 726. 21 22 25 26; 727. 1 2 16; 728. 5 6 8 26; 729. 9 10; 730. 19 20 25 26; 731. 6 13 14; 732. 3 4 5 6 16; 733. 14 16 17 18; 734. 15 16 25 26; 735. 13 14 15 16 17 18; 736. 6 9 26; 737. 14 20 26; 738. 5 6 21 22; 739. 7 8 9 10 14 15 16; 740. 15 16 17 18; 741. 19 20 25 26; 742. 6 9 10; 743. 7 8 16; 744. 13 14; 745. 7 8 25 26; 746. 6 7 8 14 16 27; 747. 1 2 9 10; 748. 14; 749. 2 9 10 26; 750. 9 10; 751. 4 7 8 23 24; 752. 13 14; 753. 16 26; 754. 9 10; 755. 9 10 11 12; 756. 16; 757. 3 4 8; 758. 2 19 20 25 26; 759. 6 16; 760. 7 8 17 18 21 22; 761. 4; 762. 9 10 26; 763. 9 10 16; 764. 16 18; 765. 19 20 25 26; 766. 3 4 6; 767. 16 17 18; 768. 9 10 16; 769. 5 6; 770. 11 12 15 16 17 18; 771. 9 10; 772. 1 2 25 26; 773. 13 14 15 16; 774. 1 2 9 14 16 22 26; 775. 5 6 15 16; 776. 1 2 8 9 10; 777. 2 19 20 25 26; 778. 3 4; 779. 6; 780. 14 16; 781. 17 18; 782. 14; 783. 9 10 19 20 26; 784. 3 4; 785. 4 6 9 27; 786. 3 4 6 7 8; 787. 14 15 16; 788. 9 10; 789. 14; 790. 9 10 16; 791. 1 2 14 17 18; 792. 9 10 16; 793. 6 16; 794. 1 2 6 13 14 16; 795. 5 6 7 8 14 26; 796. 9 26; 797. 1 2 19 20 26;

The best solution found by the EC-heuristic consists of 114 clusters (their numbers are in boldface) containing the following objects:

1: 1 59 103 106 166 180 192 207 216 222 327 332 336 358 370 399 413 424 449 462 514 747 749 776; 2: 2 63 109 121 155 181 204 220 242 299 325 390 444 450 493 503 550 577 598 620 670 690 732 766; 3: 3 15 50 209 692 707; 4: 4 203 294 386 490 509 515 591 693 727; 5: 5 308 500; 6: 6 30 38 238 290 497 522 745; 7: 7 57 60 146 164 176 187 198 302 312 318 334 342 344 354 401 418 451 476 479 526 536 546 554 579 623 646 724 757 778 784; 8: 8 119 183 194 292 365 374 375 382 414 425 468 469 486 489 538 560 597 610 695 756; 9: 9 149 256 278 300 498 544 549 769; 10: 10; 11: 11 26 31 33 34 74 83 95 167 169 173 175 196 223 298 339 347 357 383 397 410 419 441 460 484 496 504 528 537 539 567 570 611 612 613 630 640 641 648 661 691 720 725 733 735 740 767 770; 12: 12 29 89 98 134 228 262 275 291 440 453 461 495 572 658 679 706 786; 13: 13 35 279; 14: 14 42 52 54 80 97 99 100 112 113 124 147 165 199 200 215 235 248 258 305 320 328 350 359 361 398 412 422 426 428 437 466 467 472 491 510 529 545 552 574 576 584 585 588 607 636 668 704 723 729 742 750 754 762 763 768 771 788 790 792; 15: 16 19 36 67 136 148 177 244 252 396 400 448 470 475 569 627 662 721 728 795; 16: 17 46 317 487 501 507 592; 17: 18 24 28 61 72 78 90 102 104 110 122 132 135 139 151 156 162 205 230 231 232 234 255 259 261 271 277 301 307 313 314 330 362 369 371 380 381 408 429 438 447 464 474 480 481 531 558 587 593 594 595 596 600 606 608 649 651 655 656 663 681 684 697 705 709 710 730 741 758 765 777; 18: 20 174 653; 19: 21 154 157 266 746; 20: 22 55 64 76 168 218 263 457 513 581 647 674; 21: 23 82 87 217 276 282 309 310 322 323 352 378 446 456 516 602 605 619 650 702 780; 22: 25 138 267 701 714; 23: 27 93 188 191 202 286 329 494 535 578; 24: 32 56 555; 25: 37 407; 26: 39 49 287 289 416 423 439 716 772; 27: 40 241 331 360 455 523 671; 28: 41 69; 29: 43 125 247 268 761; 30: 44 62 94 101 123 126 143 197 214 274 315 388 521 527 541 586 614 621 634 635 672 687 694 698 699 731 744 752; 31: 45 179 254 345 364 385 405 406 421 463 492 517 563 583 599 604 617 639 676 743; 32: 47 51 211 221 284 333 348 483 511 633; 33: 48; 34: 53 115 321 477; 35: 58 85 127 665 764; 36: 65 304 520 542 688 734; 37: 66 478 532 559 779; 38: 68 108 152 193 249 281 285 319 337 435 571 632 660 719 753; 39: 70 224 680; 40: 71 107 246; 41: 73; 42: 75 170 195 654 659 712; 43: 77 622 791; 44: 79 141 142 172 219 239 240 265 343 356 409 430 431 436 454 473 548 616 629 643 657 773 787; 45: 81 178 367 666; 46: 84 96 201 326 394 568; 47: 86 759 793; 48: 88 120 270 273 376 417 508 689; 49: 91; 50: 92 471; 51: 105 111 306 335 644 645 711; 52: 114; 53: 116 395 547 551 675; 54: 117 296 512; 55: 118; 56: 128 708; 57: 129 601 638; 58: 130 131 171 225 303 366 384 402 553 669 703 718; 59: 133 760; 60: 137 233 346 530 683 696; 61: 140 389 415 533 556; 62: 144 253 272 420 442 726; 63: 145 212 293 297 340 355 391 580 589 794; 64: 150; 65: 153 283 368 524 573 717 739; 66: 158; 67: 159 269 502 618 755; 68: 160; 69: 161 186 206 506 543 682; 70: 163; 71: 182 311 677 722 737; 72: 184 264; 73: 185 245 341 363 443 458 562 564 626 628 748 782 789; 74: 189 226 686; 75: 190 210 227 243 280 373 387 411 465 566 673 781; 76: 208; 77: 213 445; 78: 229 351 738; 79: 236 324 403 427 433 575 582 631 664; 80: 237 488 590; 81: 250 459; 82: 251 377 561 678 736 796; 83: 257; 84: 260; 85: 288 485; 86: 295; 87: 316 404; 88: 338; 89: 349 667 700 785; 90: 353; 91: 372 482; 92: 379 393 432 783; 93: 392; 94: 434; 95: 452; 96: 499; 97: 505; 98: 518 625; 99: 519 797; 100: 525 557; 101: 534; 102: 540; 103: 565; 104: 603; 105: 609; 106: 615; 107: 624; 108: 637; 109: 642; 110: 652 751; 111: 685 775; 112: 713; 113: 715; 114: 774;

References

1. R.H. AHMADI and C.S. TANG, 1991. An Operation Partitioning Problem for Automated Assembly System Design, *Operations Research* 39, 824-835.
2. A.V. AHO, J.E. HOPCROFT and J.D. ULLMAN, 1974. *The Design and Analysis of Computer Algorithms*, Addison Wesley, Reading, Mass.
3. S.G. DE AMORIM, J.-P. BARTHÉLEMY and C.C. RIBEIRO, 1992. Clustering and Clique Partitioning: Simulated Annealing and Tabu Search Approaches, *Journal of Classification* 9, 17-41.
4. M.R. ANDERBERG, 1973. *Cluster Analysis for Applications*, Academic Press, New York.
5. E.R. BARNES, A. VANELLI and J. WALKER, 1988. A New Heuristic for Partitioning the Nodes of a Graph, *SIAM Journal on Discrete Mathematics* 1, 299-305.
6. J.-P. BARTHÉLEMY and B. MONJARDET, 1981. The Median Procedure in Cluster Analysis and Social Choice Theory, *Mathematical Social Sciences* 1, 235-267.
7. J.F. BENDERS, 1962. Partitioning Procedures for Solving Mixed Variables Programming Problems, *Numerische Mathematik* 4, 38-252.
8. T. BUI, S. CHAUDHURI, F.T. LEIGHTON and M. SIPSER, 1987. Graph Bisection Algorithms with Good Average Case Behaviour, *Combinatorica* 7, 171-191.

9. S. CHOPRA and M.R. RAO, 1991. On the Multiway Cut Polyhedron, *Networks* 21, 51-89.
10. M. DEZA, M. GRÖTSCHEL and M. LAURENT, 1992. Clique-Web Facets for Multicut Polytopes, *Mathematics of Operations Research* 17, 981-1000.
11. R. DUBES and A.K. JAIN, 1980. Clustering Methodologies in Exploratory Data Analysis, *Advances in Computers* 19, 113-228.
12. A.E. DUNLOP and B.W. KERNIGHAN, 1985. A Procedure for Placement of Standard-Cell VLSI Circuits, *IEEE Transactions on Computer Aided Design* 4, 92-98.
13. M.E. DYER and A.M. FRIEZE, 1985. On the Complexity of Partitioning Graphs Into Connected Subgraphs, *Discrete Applied Mathematics* 10, 139-153.
14. T.A. FEO and M. KHELLAF, 1990. A Class of Bounded Approximation Algorithms for Graph Partitioning, *Networks* 20, 181-195.
15. C.M. FIDUCCIA and R.M. MATTHEYES, 1982. A Linear-Time Heuristic for Improving Network Partitions, in *Proceedings of the 19th Design Automation Conference*, 175-181.
16. L.R. FORD and D.R. FULKERSON, 1974. *Flows in Networks*, 6th ed., Princeton University Press, Princeton, NJ.
17. M.R. GAREY and D.S. JOHNSON, 1979. *Computers and Intractability: A Guide to the Theory of NP-completeness*, W.H. Freeman Company, New York.
18. M.R. GAREY, D.S. JOHNSON and L. STOCKMEYER, 1976. Some Simplified NP-complete Problems, *Theoretical Computer Science* 1, 237-267.
19. F. GLOVER, 1986. Future Paths for Integer Programming and Links to Artificial Intelligence, *Computers and Operations Research* 13, 533-549.
20. F. GLOVER, 1989. Tabu-Search-Part I, *ORSA Journal on Computing* 1, 190-206.
21. F. GLOVER, 1990. Tabu-Search-Part II, *ORSA Journal on Computing* 2, 4-32.
22. F. GLOVER, 1991. Multilevel Tabu Search and Embedded Search Neighborhoods for the Traveling Salesman Problem, submitted for publication.
23. F. GLOVER, 1992. *Ejection Chains, Reference Structures, and Alternating Path Methods for the Traveling Salesman Problem*, University of Colorado, Boulder.
24. M. GRÖTSCHEL and Y. WAKABAYASHI, 1989. A Cutting-Plane Algorithm for a Clustering Problem, *Mathematical Programming* 45, 59-96.
25. M. GRÖTSCHEL and Y. WAKABAYASHI, 1990a. Facets of the Clique Partitioning polytope, *Mathematical Programming* 47, 367-387.
26. M. GRÖTSCHEL and Y. WAKABAYASHI, 1990b. Composition of Facets of the Clique Partitioning Polytope, in *Topics in Combinatorics and Graph Theory, Essays in Honour of Gerhard Ringel*, R. Bodendiek and R. Henn (eds.), Physica, Heidelberg, 277-284.
27. C.C. HAN and C.H. LEE, 1988. Comments on Mohr and Hendersons Path Consistency Algorithm, *Artificial Intelligence* 36, 125-130.
28. D.J. HAND, 1981. *Discrimination and Classification*, Wiley & Sons, New York.
29. J.A. HARTIGAN, 1975. *Clustering Algorithms*, Wiley & Sons, New York.
30. D.S. JOHNSON, C.R. ARAGON, L.A. MCGEOCH and C. SCHEVON, 1989. Optimization by Simulated Annealing: An Experimental Evaluation; Part 1, Graph Partitioning, *Operations Research* 37, 865-892.
31. E.L. JOHNSON, A. MEHROTRA and G.L. NEMHAUSER, 1991. *Min-Cut Clustering*, Georgia Institute of Technology, Atlanta.
32. D.R. JONES and M.A. BELTRAMO, 1991. Solving Partitioning Problems with Genetic Algorithms, in *Proceedings of the 4th International Conference on Genetic Algorithms*, 442-449.
33. A.B. KAHNG, 1989. Fast Hypergraph Partition, in *Proceedings of the 26th Design Automation Conference*, 762-766.
34. B.W. KERNIGHAN, 1971. Optimal Sequential Partitions of Graphs, *Journal of the ACM* 18, 34-40.
35. B.W. KERNIGHAN and S. LIN, 1970. An Efficient Heuristic Procedure for Partitioning Graphs, *The Bell System Technical Journal* 49, 291-307.
36. B. KRISHNAMURTI, 1984. An Improved Min-Cut Algorithm for Partitioning VLSI Networks, *IEEE Transactions on Computers* C-33, 438-446.
37. K.R. KUMAR, A. KUSIAK and A. VANNELLI, 1986. Grouping Parts and Components in Flexible Manufacturing Systems, *European Journal on Operational Research* 24, 387-397.
38. T. LENGAUER, 1990. *Combinatorial Algorithms for Integrated Circuit Layout*, Teubner, Stuttgart, and Wiley & Sons, Chichester.
39. S. LIN and B.W. KERNIGHAN, 1973. An Effective Heuristic Algorithm for the Traveling Salesman Problem, *Operations Research* 21, 498-516.
40. J.A. LUKES, 1975. Combinatorial Solution to the Partitioning of General Graphs, *IBM Journal of Research and Development* 19, 170-180.
41. F. MARCOTORCHINO and P. MICHAUD, 1980. Optimisation en Analyse des Données. Relationelle, in *Data Analysis and Informatics*, E. Diday et al. (eds.), North-Holland, 655-670.
42. F. MARCOTORCHINO and P. MICHAUD, 1980. Optimization in Exploratory Data Analysis (Preference and Similarity Aggregation), *Methods of OR* 40, 169-173.
43. F. MARCOTORCHINO and P. MICHAUD, 1981. Heuristic Approach of the Similarity Aggregation Problem, *Methods of OR* 43, 395-404.
44. M. MINOUX and E. PINSON, 1987. Lower Bounds to the Graph Partitioning Problem Through Generalized Linear Programming and Network Flows, *R. A. I. R. O. Recherche Operationelle / Operations Research* 21, 349-364.
45. H. MÜHLENBEIN, M. GORGES-SCHLEUTER and O. KRÄMER, 1988. Evolution Algorithms in Combinatorial Optimization, *Parallel Computing* 7, 65-88.
46. O. OPITZ and M. SCHADER, 1984. Analyse Qualitativer Daten: Einführung und Übersicht, *OR Spektrum* 6, 67-83, 133-140.
47. C.H. PAPADIMITRIOU and K. STEIGLITZ, 1982. *Combinatorial Optimization: Algorithms and Complexity*, Prentice-Hall, Englewood Cliffs, NJ.
48. S. RÈGNIER, 1965. Sur Quelques Aspects Mathématiques des Problèmes de Classification Automatique, *I.C.C. Bulletin* 4, 175-191.
49. C. ROUCAIROL and P. HANSEN, 1987. Problème de la Bipartition Minimale d'un Graphe, *R. A. I. R. O. Recherche Operationelle / Operations Research* 21, 325-348.
50. L.A. SANCHIS, 1989. Multiple-Way Network Partitioning, *IEEE Transactions on Computers* C-38, 62-81.
51. M. SCHADER and U. TÜSHAUS, 1985. Ein Subgradientenverfahren zur Klassifikation Qualitativer Daten, *OR Spektrum* 7, 1-5.
52. H. SPÄTH, 1985. *Cluster Dissection and Analysis: Theory, FORTRAN Programs, Examples*, Wiley & Sons, Chichester.
53. U. TÜSHAUS, 1983. Aggregation Binärer Relationen in der Qualitativen Datenanalyse, *Mathematical Systems in Economics* 82, Hain, Königstein.
54. Y. WAKABAYASHI, 1986. *Aggregation of Binary Relations: Algorithmic and Polyhedral Investigations*, Dissertation, Universität Augsburg.
55. C.T. ZAHN, 1964. Approximating Symmetric Relations by Equivalence Relations, *SIAM Journal on Applied Mathematics* 12, 840-847.