Mathematical Programming 81 (1998) 229–256

The node capacitated graph partitioning problem: A computational study

C.E. Ferreira [a,*], A. Martin [b], C.C. de Souza [c], R. Weismantel [b], L.A. Wolsey [d]

[a] *Universidade de São Paulo, CP Rua Monte Alegre 984, Perdiz, 05014 Sao Paulo, Brazil*
[b] *Konrad-Zuse-Zentrum für Informationstechnik Berlin, Germany*
[c] *Universidade Estadual de Campinas, Brazil*
[d] *CORE, Université Catholique de Louvain, Belgium*

Received 12 October 1994; revised manuscript received 30 January 1996

Abstract

In this paper we consider the problem of k-partitioning the nodes of a graph with capacity restrictions on the sum of the node weights in each subset of the partition, and the objective of minimizing the sum of the costs of the edges between the subsets of the partition. Based on a study of valid inequalities, we present a variety of separation heuristics for cycle, cycle with ears, knapsack tree and path-block cycle inequalities among others. The separation heuristics, plus primal heuristics, have been implemented in a branch-and-cut routine using a formulation including variables for the edges with nonzero costs and node partition variables. Results are presented for three classes of problems: equipartitioning problems arising in finite element methods and partitioning problems associated with electronic circuit layout and compiler design. © 1998 The Mathematical Programming Society, Inc. Published by Elsevier Science B.V.

Keywords: Branch-and-cut algorithm; Clustering; Compiler design; Equipartitioning; Finite element method; Graph partitioning; Layout of electronic circuits; Separation heuristics

1. Introduction

Graph partitioning problems arise often when graphs are used as a modeling tool. The particular class of partitioning problems that is considered here involves partitioning the node set of a graph so that the sum of node weights within each set of the partition is limited, and the objective is to minimize the sum of the costs of edges having their endpoints in different sets of the partition (the associated multicut). This problem can perhaps be viewed as the prototype for breaking up a graph into smaller pieces where the edge weights w_{ij} measure the importance of considering the nodes i and j together in the same subgraph.

* Corresponding author.

Such problems are usually tackled heuristically. A recent study of Johnson et al. [13] involves extensive testing and comparison of simulated annealing and a successful deterministic heuristic due to Kernighan and Lin [16] on graphs with 500 nodes and upwards. The paper [13] also contains a large number of references to earlier works. The results on the exact solution of such problems, are however, sparse. A recent paper of Johnson et al. [14] presents a heuristic column generation algorithm that terminates with upper and lower bounds on the optimal value, and an optimality proof if the two are equal. For a set of compiler design problems involving graphs with between 31 and 61 nodes discussed further below, nine out of 12 instances are solved to optimality. Holm and Sorensen [12] have implemented a branch-and-cut (BC) algorithm for the node capacitated graph partitioning problem that is based on simple cuts and on reducing the symmetry of the solutions in order to keep the size of the branch-and-bound tree as small as possible. The algorithm has been tested on randomly generated graphs and the authors proved optimality for instances up to 20 nodes.

Various special cases or closely related problems such as the equipartitioning problem, the max cut problem, and the equivalent problem of optimizing a quadratic function in 0/1 variables have received a lot of attention. Based largely upon polyhedral studies of the convex hull of incidence vectors of these problems, computational results for cut problems on special graphs with up to 1600 nodes are presented in [3], and equipartitioning problems on complete graphs with up to 70 nodes have been tackled, see [5,6].

These works, plus the three applications discussed below, motivated us to study the polyhedral structure of the node capacitated graph partitioning problem. Formally, given a graph $G = (V,E)$, an integer $K \in \mathbb{N}$, edge weights c_e for $e \in E$, node weights f_i *for* $i \in V$ and a capacity F, the problem is to find a partition $(V_1, V_2, \ldots, V_m)$ of V with $m \leqslant K$, $\sum_{i\in V_j} f_i \leqslant F$ for $j = 1, \ldots, m$ and such that

$$\sum_{e\in\delta(V_1,V_2,\ldots,V_m)} c_e$$

is minimized, where $\delta(V_1,V_2,\ldots,V_m)$ denotes the set of edges whose endnodes belong to different elements of the partition, typically called a *multicut*. When $m = 2$, we also use the notation $\delta(V_1, V_2) = \{e = (i,j) \colon i \in V_1, j \in V_2\}$ even if $V_1 \cup V_2 \subset V$. If $V_1 \cup V_2 = V$ we also use $\delta(V_1)$ in place of $\delta(V_1, V \setminus V_1)$. For convenience, we introduce the term *cluster*. A cluster is a subset of V (possibly empty). A *cluster partition* $(V_1, V_2, \ldots, V_K)$ into K disjoint clusters ($V_i \cap V_j = \emptyset$ for all i different from j, $\cup_{i=1}^{K} V_i = V$, $V_i \subseteq V$ for all i) corresponds to a partition $(V_1, V_2, \ldots, V_m)$ of V, $1 \leqslant m \leqslant K$, where $V_1, V_2, \ldots, V_m$ are the nonempty clusters in the cluster partition.

The theoretical results obtained are discussed in a companion paper [8], whereas here we report on our experience attempting to incorporate the families of valid inequalities discovered into a branch-and-cut system to solve the three classes of problem instances.

The outline of the paper is as follows. In Section 2 we discuss the three classes of applications from which our test problems are drawn. In Section 3 we briefly describe the formulation and the inequalities to be used. For most of the inequalities the separation problem is NP-complete and we resort to heuristic ideas in order to generate different families of valid inequalities. In Section 4 we present these heuristics in some detail. Because of the limited literature on separation and because these heuristics involve very common structures such as trees, cycles and minimal infeasible sets, we feel that such details are worth reporting. In Section 5 we discuss other details of the branch-and-cut implementation such as primal heuristics and branching rules, and finally in Section 6 we present our computational results.

2. Applications

In this paper we deal with instances of graph partitioning problems arising from three different applications: compiler design, finite element computations associated with meshes, and partitioning problems coming from the design of electronic circuits.

The compiler design application is described in [14]. A compiler consists of several modules, where each module is a set of procedures or subroutines with a corresponding memory requirement. The modules must be combined to form clusters whose size is restricted by storage capacity. Modules assigned to different clusters cause high communication costs, because such communications might require memory swapping. The objective of the compiler designer is to assign the modules to clusters so as to meet the storage requirements and to minimize the total communication costs between modules in different clusters. Representing the modules as nodes of a graph and the communication between modules as edges of this graph, the problem translates into a node capacitated graph partitioning problem as pointed out in [14].

Another application involves the use of parallelism in Finite Element computations. Typically in a Finite Element mesh each element is associated to a set of variables and equations of a linear system. The central question is how to solve such a system efficiently when several processors are available to execute that task. It is known that to have an efficient parallel algorithm one has to balance the workload and to minimize the communication among the processors. In terms of the Finite Element problem above, this corresponds to partitioning the mesh into K (the number of processors) subregions containing several elements and having small "boundaries" between these subregions. Roughly speaking the workload on the processors depends upon the number of elements (variables and equations) associated to it. It is reasonable then to try to assign the same (or almost the same) number of elements to each processor. Thus, if n is the number of elements in the mesh, the problem to be solved is that of finding a partition of the mesh into K subregions containing at most $\lceil n/K \rceil$ elements and minimizing the "boundaries". Such a problem can be modeled as a graph partitioning problem (see [20,21] for more details).

Finally, the node capacitated graph partitioning problem appears as a subproblem in the layout of electronic circuits (see [15,23]). Given a set of cells where each cell i covers an area f_i, a ground area (also called the master) and a list of nets specifying the connections between cells so as to realize a certain logic function, the layout problem is to find an assignment of the cells to locations on the master and to route the nets by wires so that certain side constraints are met and an objective function, such as the total wiring length, is minimized. The side constraints include, for instance, that no two cells must overlap, that certain wires must not exceed a given length, or restrictions on the feasible locations for certain cells. A common approach used to tackle this problem is to apply decomposition techniques (see [17,23]). One such approach involves the problem of grouping cells into clusters filling at most a prespecified area so that the number of nets connecting cells of different clusters is as small as possible. Introducing a graph $G = (V, E)$ whose nodes represent the cells and where an edge (i, j) indicates that cells i and j must be connected by a wire, we obtain a node capacitated graph partitioning problem.

These three classes of instances differ considerably with respect to the distribution of the node weights, the knapsack capacity relative to the total node weight, and the structure of the graphs. In the compiler design instances the graphs are sparse and there are just a few nodes having both a very large weight and a high degree. For the mesh problems the underlying graphs are again sparse and the node degrees are at most 4. All the node weights are 1, and the objective is to find an equicut in the graph of minimum size. The instances arising from VLSI-design involve a small number of different node weights. Typically, about 50% of the items have weight 1, 30% of the items have weight 2 and the node weights of the remaining 20% of the items lie in the range from 3 up to 100. The graphs are sparse, yet without any well-defined structure.

3. Polyhedral background: A short survey

In this section we briefly present the formulation and the valid inequalities we use to tackle the node capacitated graph partitioning problem. For the proofs and a more detailed discussion, we refer to the companion paper [8].

Introducing variables z_i^k, $i \in V$, $k = 1, \ldots, K$ and y_e, $e \in E$ such that $z_i^k = 1$ if node i lies in cluster k and $z_i^k = 0$ otherwise and $y_e = 1$ if e lies in the multicut $\delta(V_1, \ldots, V_K)$ and $y_e = 0$ otherwise, we obtain a formulation

$$
\text{(P)} \quad
\begin{array}{lll}
z = \min & \sum\limits_{e \in E} c_e y_e & \\
& \sum\limits_{k=1}^{K} z_i^k = 1 & \text{for all } i \in V, \\
& \sum\limits_{i \in V} f_i z_i^k \leqslant F & \text{for all } k = 1, \ldots, K, \\
& y_{ij} \geqslant \sum\limits_{k \in K_1} z_i^k - \sum\limits_{k \in K_1} z_j^k & \text{for } K_1 \subseteq \{1, \ldots, K\},\ (i,j) \in E, \\
& y_{ij}, z_i^k \in \{0, 1\} & \text{for all } (i,j) \in E, \\
& & \text{for all } i \in V \text{and } k = 1, \ldots, K.
\end{array}
$$

Observing that the z_i^k variables do not appear in the objective function, the next step is to consider the model

$$(\mathrm{P}') \qquad z = \min \sum_{e \in E} c_e y_e, \quad y \in Y,$$

where $Y \in \mathbb{R}^E$ is the projection onto the y-space of the feasible region of model (P).

In [8] we have introduced several classes of valid, and under certain assumptions, facet-defining inequalities for the polytope $P^{K,F}(G) := \mathrm{conv}(Y)$. Though this polytope is a natural one to study, there is a potentially important drawback: given an edge subset $E' \subseteq E$, it is NP-hard to verify whether its associated incidence vector $\chi^{E'} \in Y$. In order to guarantee feasibility of the solution, we need to incorporate the number of clusters and the variables z_i^k. As every inequality valid for $P^{K,F}(G)$ is also valid for the polytope associated with model (P), we can use the valid inequalities for $P^{K,F}(G)$ in a branch-and-cut algorithm applied to formulation (P).

The well-known triangle and cycle inequalities for cut polytopes [2,10] turned out to be ineffective in tackling the node capacitated partitioning problem. Below we list some of the valid inequalities for $P^{K,F}(G)$ that are specific to the latter problem [8] and were tested in our computations.

First we present some definitions. A node set S forms a *q-cover* if $\sum_{i \in S} f_i > qF$. The q-cover is said to be *minimal* if no proper subset of it is a q-cover. A 1-cover is just called a *cover*. A graph (S, E_S) is a *cycle with tails*, if it consists of a cycle plus a set of one or more paths each starting at some node of the cycle, and otherwise disjoint both among themselves and from the cycle, i.e. if there exists a cycle (C, E_C), $C \subseteq S$ and paths $P_1, \dots, P_t$ such that $V(P_i) \cap C = \{u_i\}$ for $i = 1, \dots, t$ and $V(P_i) \cap V(P_j) \cap (S \setminus C) = \emptyset$, $(i, j \in \{1, \dots, t\}, i \neq j)$ and $(C \cup V(P_1) \cup \cdots \cup V(P_t)$, $E_C \cup E(P_1) \cup \cdots \cup E(P_t)) = (S, E_S)$. Next consider a graph $G = (V, E)$. An *ear decomposition* of G is a decomposition

$$G = C \cup P_1 \cup \cdots \cup P_r,$$

where C is a cycle and P_{i+1} is a path whose endnodes belong to $C \cup P_1 \cup \cdots \cup P_i$, but whose inner nodes do not. The path P_i is called an *ear* and the ear decomposition is said to be *nondegenerate* if the two endnodes of P_i are distinct for all ears $i = 1, \dots, r$.

Proposition 3.1 ((Multiple) cover inequalities). *Let a graph $G = (V, E)$ be given and assume that $S \subseteq V$ is a q-cover.*

(i) *If the subgraph (S, E_S) is a tree, the q-cover tree inequality*

$$\sum_{e \in E_S} y_e \geqslant q$$

is valid for $P^{K,F}(G)$.

(ii) *If the subgraph (S, E_S) is a cycle, the q-cover cycle inequality*

$$\sum_{e \in E_S} y_e \geqslant q + 1$$

is valid for $P^{K,F}(G)$.

(iii) *If the subgraph* (S, E_S) *has an ear decomposition, the q-cover cycle with ear inequality*

$$\sum_{e \in E_S} y_e \geqslant q+1$$

is valid for $P^{K,F}(G)$.

(iv) *If the subgraph* (S, E_S) *is a cycle with tails where* (C, E_C) *denotes the cycle in* (S, E_S), *the q-cover cycle with tails inequality*

$$\sum_{e \in E_C} y_e + 2 \sum_{e \in E_S \setminus E_C} y_e \geqslant q+1$$

is valid for $P^{K,F}(G)$.

When the tree subgraph is a *star* with nodes in $S = \{r\} \cup N(r)$, $E_S = \delta(r)$ where $N(r) := \{u \in V\colon (r,u) \in \delta(r)\}$ and when S forms a 1-cover, there is an easy way of strengthening the tree inequality.

Proposition 3.2 (Star inequality). *If the subgraph* (S, E_S) *is a star and S is a cover, the inequality*

$$\sum_{e \in E_S} y_e \geqslant |N(r)| - n_r$$

is valid for $P^{K,F}(G)$, *where* $n_r = \max\{|U|\colon U \subseteq N(r), \sum_{j \in U} f_j \leqslant F - f_r\}$.

The cycle with ear inequality can also often be strengthened. We consider the case of $q = 1$. Given a particular nondegenerate ear decomposition $C \cup P_1 \cup \cdots \cup P_r$ of G_S, let u_i, v_i be the distinct endnodes of P_i, for $i = 1, \ldots, r$, and $A := \{e \in E\colon e = (u_i, v_i) \text{ for some } i\}$ and g_e be the number of occurrences of e in the list $\{(u_1, v_1), \ldots, (u_r, v_r)\}$.

Proposition 3.3 (Cycle with ear inequality). *If* (S, E_S) *is a subgraph of* $G = (V, E)$ *that has an ear decomposition and S is a cover, then the inequality*

$$\sum_{e \in E_S} y_e - \sum_{e \in E \cap A} g_e y_e \geqslant 2$$

is valid for $P^{K,F}(G)$.

To define the next family of inequalities, we suppose a set $X = \{x_1, \ldots, x_t\}$ of nodes is given with $t \geqslant 2$, and $r \geqslant 2$ node sets $C_1, C_2, \ldots, C_r$ inducing cycles where $C_i \cap C_j = X$ for all $i, j \in \{1, \ldots, r\}$. Thus each cycle C_j decomposes into t subpaths $\{P_{ij}\}_{i=1}^{t}$ starting at x_i, followed by a possibly empty node set Q_{ij}, and terminating at $x_{(i+1)(\mathrm{mod}\, t)}$. The weight of a node set Q_{ij} is simply the sum of the weights of the nodes in Q_{ij}. The sets $Q^1, Q^2, \ldots, Q^k$, for $1 \leqslant k \leqslant tr$, are defined to be the k distinct node sets $\{Q_{ij}\}$ of largest weight. Setting $S := \bigcup_{j=1}^{r} C_j$ and $E_S := \bigcup_{j=1}^{r} E(C_j)$, the graph (S, E_S) is called a *path-block cycle*. An example of such a graph is illustrated in Fig. 1.

Proposition 3.4 (Path-block cycle inequality). *If (S, E_S) is a subgraph of $G = (V, E)$ that is a path-block cycle, the inequality*

$$\sum_{j=1}^{r} \sum_{e \in E(C_j)} y_e \geqslant 2r$$

is valid for $P^{K,F}(G)$ if and only if $S \setminus (\bigcup_{j=1}^{r-1} Q^j)$ is a cover.

For the example in Fig. 1, if $F = 16$ and $f_i = 1$ for all $i \in V$, the path-block cycle inequality is given by

$$\begin{aligned} &y_{1,19} + y_{1,16} + y_{1,21} + y_{1,6} + y_{1,12} + y_{1,17} + y_{2,7} + y_{2,12} \\ &+ y_{2,17} + y_{2,3} + y_{2,13} + y_{2,18} + y_{3,13} + y_{3,18} + y_{3,8} + 2y_{3,4} \\ &+ y_{4,8} + y_{4,9} + y_{4,11} + y_{4,15} + y_{5,9} + y_{5,10} + y_{5,14} + y_{5,20} \\ &+ y_{5,16} + y_{5,19} + y_{6,7} + y_{14,15} + y_{10,11} + y_{20,21} \geqslant 6. \end{aligned}$$

Finally we introduce the class of knapsack tree inequalities. Let $\sum_{i \in V} a_i x_i \leqslant a_0$ be a valid inequality for the knapsack polytope conv$\{x \in \mathbb{R}^V \colon \sum_{i \in V} f_i x_i \leqslant F,\ x_i \in \{0,1\},\ i \in V\}$, with $a_i \geqslant 0$ for all $i \in V$. We choose a node r, and a subtree (T, E_T) of G rooted at r. For $i \in T$, let P_i denote the edge set of the path joining i to the root r in (T, E_T). Moreover, $p(j)$ is the (unique) predecessor of node j on the path P_j.

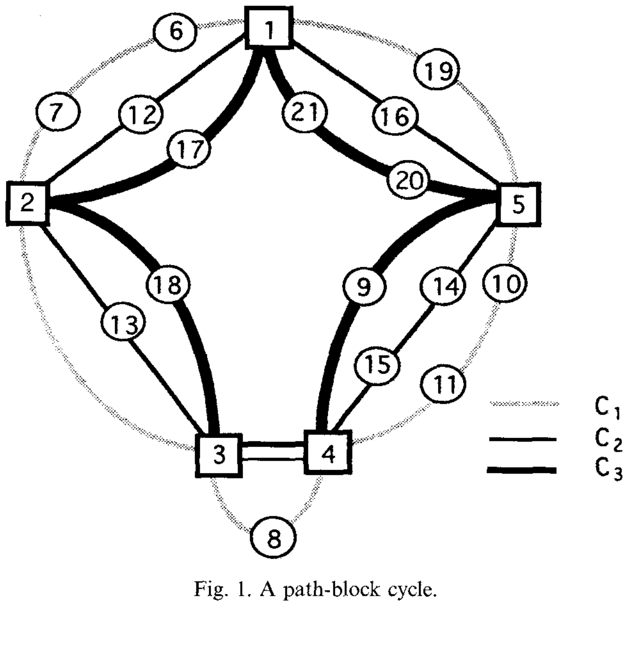

Fig. 1. A path-block cycle.

Proposition 3.5 (Knapsack tree inequality). *If $\sum_{i\in V} a_i x_i \leqslant a_0$ is a valid inequality for the knapsack polytope* $\operatorname{conv}\{x \in \mathbb{R}^V\colon \sum_{i\in V} f_i x_i \leqslant F,\ x_i \in \{0,1\}, i \in V\}$, *with $a_i \geqslant 0$ for all $i \in V$, then the inequalities*

$$\sum_{i\in T} a_i \left(1 - \sum_{e\in P_i} y_e\right) \leqslant a_0$$

and

$$\sum_{i\in T} \min\{\alpha_i, \alpha_0, a_0\} y_{p(i),i} \geqslant \alpha_0$$

are valid for $P^{K,F}(G)$, where $\alpha_i := \sum_{\{j: i\in V(P_j)\}} a_j$ and $\alpha_0 := \sum_{i\in T} a_i - a_0$.

In [8] we also examine how the inequalities presented above can be strengthened, and if and when the resulting inequalities define facets. This information is put to use in Section 4 in which we describe the routines we have developed to generate these inequalities as cutting planes.

4. Separation and strengthening of inequalities

Having presented several inequalities valid for the polytope $P^{K,F}(G)$ in the previous section, we now discuss how to use them in a branch-and-cut algorithm. Here, the key issue is separation. For a class $\mathscr{I}$ of inequalities for problem (P), the *separation problem* is defined as follows:

Given $(z', y') \in \mathbb{R}^{K|V|+|E|}$, decide whether (z', y') satisfies all inequalities in $\mathscr{I}$ and, if not, find an inequality in $\mathscr{I}$ violated by (z', y').

This issue is now addressed for the different classes of inequalities introduced in Section 3. Note that various separation algorithms for cut and multicut polyhedra have been developed involving trees and cycles in [2,4,7,11]. However, the ideas behind them differ significantly from those presented here because of the node partitioning constraints. Below we assume that (z', y') is the *current fractional* solution that we would like to cut off.

4.1. Trees and stars

It can be shown that the separation problem for the (1-cover) tree inequalities is NP-hard by reduction from the Steiner Tree Problem [20]. Thus, we concentrate on heuristic procedures for the separation of tree inequalities.

The main difficulty in finding violated tree inequalities lies in the fact that two conflicting objectives need to be taken into account. On the one hand we look for a tree whose sum of edge weights (weights given by $y \in \mathbb{R}^E$) is as small as possible, while on the other hand the nodes of the tree must form a cover. The first objective calls for small trees, and the second for large trees. Our heuristic tries to play on both objectives so as to find violated tree inequalities. We proceed iteratively. Each iteration begins with a tree T such that $\sum_{e \in E_T} y'_e < 1$, i.e., it is a "potentially" violated tree inequality. If the set V_T is a cover, then T induces a violated tree inequality and we stop. Otherwise we try to enlarge this tree, selecting among all nodes $j \in V \setminus V_T$ for which there exists an edge $(i,j) \in E$ with $i \in V_T$, one with minimum *weighted distance*, i.e., we compute $j^* :=$ argmin $\{(y'_{(i,j)}/f_j)\colon i \in V_T\}$ and add node j^* and edge (i,j^*) to the tree. If $\sum_{f \in E_T \cup (i,j^*)} y'_f \geqslant 1$ the procedure fails, otherwise we repeat the above steps with the tree $T + (i,j^*)$.

In our implementation, this procedure is called several times using different isolated nodes as initial tree. The same procedure can be used for the separation of q-cover tree inequalities by replacing the value of the cluster capacity F by qF. Now we discuss a way to strengthen the tree inequalities. Let T be a tree subgraph of G, V_T a cover and $\sum_{e \in E_T} 2y_e \geqslant 2$ the corresponding tree inequality multiplied by 2. Let $A := E(V_T) \setminus E_T$, and consider an edge $(i,j) \in A$ (see Fig. 2).

Let C be the fundamental cycle contained in $T + e$. It is not difficult to see that the inequality

$$\sum_{e \in E_T \setminus E_C} 2y_e + \sum_{e \in E_C} y_e \geqslant 2$$

is valid for $P^{K,F}(G)$ (it is the same as a cycle with tails inequality), and all points satisfying the tree inequality at equality also satisfy this inequality at equality. Thus in some sense this inequality is a strengthening of the initial tree inequality. Similar ideas can be used to iteratively bring in other edges of A. Of course, the resulting inequality depends on the order in which we consider the edges. Our code includes two strengthening procedures that we now discuss in more detail.

The first works as follows. Suppose that, in some iteration of the strengthening procedure, T is the current support of the inequality and $A := E(V_T) \setminus E_T$. For every edge $e = (i,j) \in A$, let P_e be a shortest path in T between i and j with edge lengths y'. If P_e contains more than one edge whose coefficient is equal to one in the current inequality, then we delete edge e from A. Having performed this step, the updated set $A \subseteq E(V_T) \setminus E_T$ consists of all edges (i,j) such that the shortest path between i and j

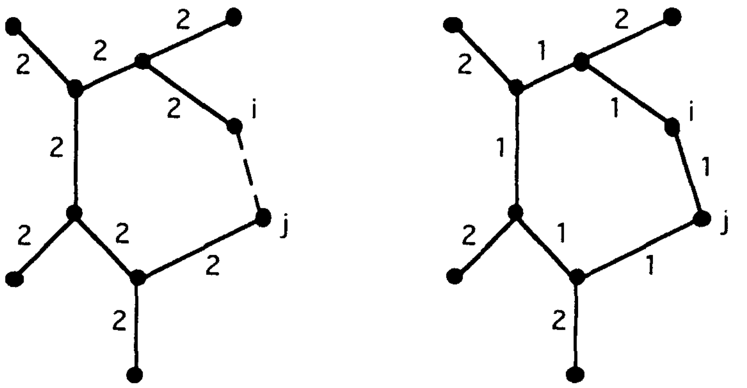

Fig. 2. Strengthening of the tree inequality.

contains only edges with coefficient 2, with at most one exception. Next, we determine $e^* \in A$ such that $\sum_{e \in P_{e^*}} y'_e - y'_{e^*}$ is maximum. A new inequality is then obtained by reducing coefficients of the variables y_e by one for all $e \in P_{e^*}$ and setting the coefficient of variable y_{e^*} to one. The face defined by the new inequality in $P^{K,F}(G)$ is of higher dimension than the previous one, since all points in the old face lie on the hyperplane $\sum_{e \in P_{e^*}} y_e = y_{e^*}$. The edge e^* is added to T and a new iteration is started.

The second strengthening procedure tries to create an inequality with a lower left hand side value than the original tree inequality, though here the dimension of the corresponding face does not necessarily increase. Initially we sort the edges of A in decreasing order according to the gain if the edge e is added to the tree, i.e., let $(i,j) \in A$ and P_{ij} be the shortest path between i and j in the support where the value of the current LP solution y'_e is interpreted as the cost of an edge e. The gain from adding edge (i,j) to the tree is $\sum_{e \in P_{ij}} y'_e - y'_{(i,j)}$. We consider the edges in A in this fixed order, though after some iterations the value $\sum_{e \in P_{ij}} y'_e - y'_{(i,j)}$ will no longer correspond to the actual gain when adding (i,j) to the graph induced by the current inequality.

Each iteration starts by eliminating from A the edges which are chords of some cycle in the current support graph. Suppose that $\pi y \geqslant 2$ is the current inequality. Let e be the next edge in A that is being considered for possible addition to the structure and C be the smallest cycle that e creates in the support graph. The coefficient of edge e is set to one. The coefficients of the remaining edges in C are updated as follows.

If C has at most one edge in common with other cycles in the support, then these coefficients are all reduced by one (see Fig. 3). If C has more than one edge in common with other cycles in the support, the coefficients of the edges that do not belong to other cycles in the support are reduced by one while the others remain unchanged. The coefficients of the latter edges cannot be reduced because then the resulting graph would be disconnected and the inequality would no longer be valid. This situation is illustrated in Fig. 4 where we assume that $f_i = 1$ for all $i \in V$ and $F = 10$. If edge e is added and we reduce the coefficients for all edges in the cycle C by one, node i is no longer connected in the resulting support graph and since the remaining nodes

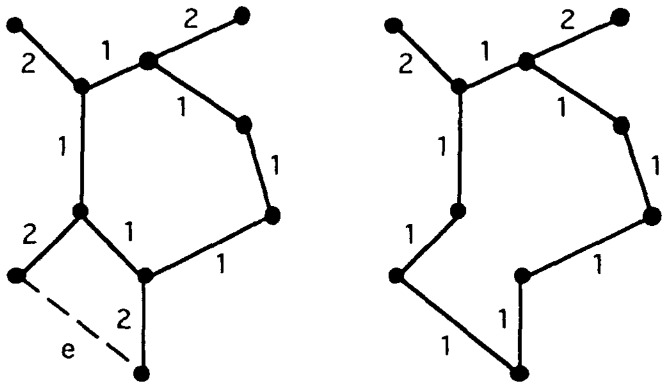

Fig. 3. Strengthening of the tree inequality.

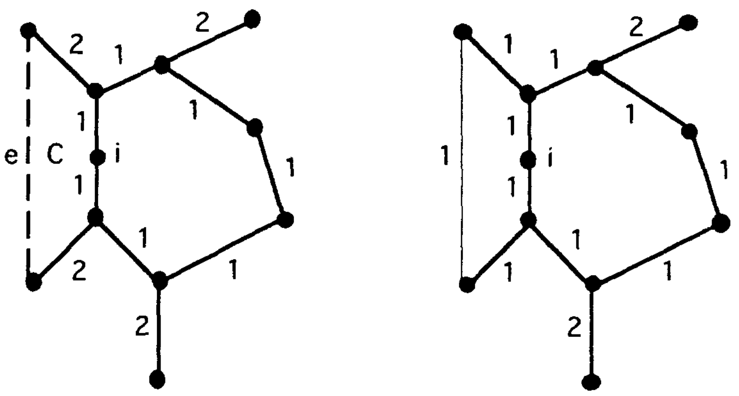

Fig. 4. Strengthening of the tree inequality.

no longer form a cover, the resulting inequality is not valid. Let π' be the support vector of the inequality obtained after changing the coefficients as explained above. If $\pi'y' < \pi y'$, then we replace π by π' and iterate. At the end of the strengthening procedure, the support of the final inequality is either a tree or a cycle with ears and tails and the corresponding node set a cover.

We have also implemented a separation routine for the star inequalities, see Proposition 3.2. Often one obtains a more violated star inequality by choosing a proper subset of the edges incident to the root node r. As an illustration, consider the following example. Suppose $F = 10$, $r = 1$, $f_1 = 5$ and the neighbors of r are $\{2, 3, 4, 5, 6, 7\}$ with weights $1, 2, 2, 4, 4, 4$, respectively. Here, $n_r = 3$ and $|N(r)| = 6$. Considering only the edges $(1, 4)$, $(1, 5)$, $(1, 6)$ and $(1, 7)$ we obtain $n_r = 1$ and $|N(r)| = 4$. Hence, the left hand side of the original star inequality is reduced by the value $y_{1,2} + y_{1,3}$ and the right hand side remains unchanged. We have implemented a routine for this "strengthening". In addition, our code tries to find violated star inequalities not only in the original graph but also in the graph obtained by contracting certain edges. This contraction yields nodes of higher weights that help in finding other violated star inequalities. Suppose, we have contracted one edge $(i, j) \in E$ and the corresponding star inequality is $\sum_{e \in \delta(i) \cup \delta(j) \setminus \{(i,j)\}} y_e \geqslant \alpha$. In order to obtain a valid inequality for the original graph, we need to calculate a lifting coefficient for this edge (i, j). So suppose that the edge (i, j) is cut in the original graph. Then we obtain two stars in the graph with edge set $E \setminus \{(i, j)\}$: one star is rooted at node i; the other star is rooted at node j. The value $|\delta(i) \setminus \{(i, j)\}| - n_i$ represents the right hand side of the star inequality with root i in the graph with edge set $E \setminus \{(i, j)\}$. Correspondingly, $|\delta(j) \setminus \{(i, j)\}| - n_j$ is the right hand side of the star inequality with root j in the graph with edge set $E \setminus \{(i, j)\}$. Therefore, $\alpha - ((|\delta(i) \setminus \{(i, j)\}| - n_i) + (|\delta(j) \setminus \{(i, j)\}| - n_j))$ is the correct lifting coefficient for the edge (i, j) and the lifted star inequality reads as

$$\sum_{e \in \delta(i) \cup \delta(j) \setminus \{(i,j)\}} y_e + [\alpha - ((|\delta(i) \setminus \{(i, j)\}| - n_i) + (|\delta(j) \setminus \{(i, j)\}| - n_j))] y_{(i,j)} \geqslant \alpha.$$

It is straightforward to derive a polynomial time algorithm in order to compute this lifting coefficient.

In our implementation we construct a sequence of edges that contains all edges whose value in the current LP solution is below a given threshold parameter. All the edges in this sequence are then contracted. For the first edge in this sequence we calculate the lifting coefficient according to the above formula. For the remaining edges in the sequence we choose the right hand side of the star inequality as the corresponding coefficient. This provides certainly an upper bound on the exact value of this coefficient and hence yields a valid lifted inequality.

4.2. *Cycles and cycles with ears*

One can show that the separation problem for the (1-cover) cycle inequalities is NP-hard by reduction from the Traveling Salesman Problem [20]. Hence, we concentrate again on heuristic separation routines for cycle inequalities. We have implemented two different heuristic ideas to find violated cycle inequalities. Below these are referred to as *Cycles 1* and *Cycles 2*.

The *Cycles 1* routine is executed for each edge $e = (i, j)$ in the graph and is based on shortest path computations. It works as follows. For every $(i, j) \in E$, we determine a shortest path P between i and j in $(V, E \setminus (i, j))$ using as costs on the edges the value of the current LP solution y'. If no such path exists the procedure fails, and otherwise we check if $\sum_{e \in E_P \cup \{(i,j)\}} y'_e < 2$. In this case we have a "potentially violated" cycle inequality. If in addition $V(P)$ is a cover, the cycle $P + (i, j)$ induces a violated cycle inequality. Otherwise $V(P)$ is not a cover and we try to extend the cycle $P + (i, j)$ by adding ears as explained after the presentation of the Cycles 2 procedure.

The idea in Procedure *Cycles 2* is to build a rooted tree and to find a violated cycle inequality when an edge is added to the tree. If no violated cycle inequality is produced, the tree is perturbed to produce a new tree with heavier paths hanging from the root. After perturbation, the initial step is repeated. More details are given below.

Procedure Cycles 2 is executed for each node in the graph and works as follows. For every node i_0 in V, a tree T is built as described in the tree separation routine. Let A be the set of edges e in $E(V_T) \setminus E_T$ that induce a cycle (C, E_C) in $T + e$ with $\sum_{e \in E_C} y'_e < 2$ and $i_0 \in C$. If there exists an edge $e \in A$ whose induced cycle is a cover, then a violated cycle inequality is found and we stop. Otherwise, let e^* be the edge in A and $(C_{e^*}, E_{C_{e^*}})$ the induced cycle for which $\sum_{u \in C_{e^*}} f_u$ is maximized. We order the edges in $E_{C_{e^*}} \setminus \{e^*\}$ according to decreasing values y'. Let $\tilde{e}$ be the next edge with respect to this ordering. We define a new tree $T(\tilde{e})$ by setting $T(\tilde{e}) := T + e^* - \tilde{e}$. Let $A(\tilde{e})$ be the set of edges e' in $E(V_{T(\tilde{e})}) \setminus E_{T(\tilde{e})}$ whose induced cycle $C_{e'}$ in $T(\tilde{e}) + e'$ satisfies $\sum_{e \in E_{C_{e'}}} y'_e < 2$ and $i_0 \in V_{C_{e'}}$. If there exists an edge $e' \in A(\tilde{e})$ such that $C_{e'}$ satisfies $\sum_{u \in V_{C_{e'}}} f_u > F$, then a violated cycle inequality has been found. Otherwise, let e' be the edge for which $\sum_{u \in V_{C_{e'}}} f_u$ is maximum among all edges in $A(\tilde{e})$. If $\sum_{u \in V_{C_{e'}}} f_u > \sum_{u \in V_{C_{e^*}}} f_u$, then a new iteration is started after replacing T, e^* and C_{e^*} by $T(\tilde{e})$, e' and $C_{e'}$, respectively. If, for all $\tilde{e}$ in C_{e^*} and for all $e' \in A(\tilde{e})$, we have that $\sum_{u \in V_{C_{e'}}} f_u \leqslant \sum_{u \in V_{C_{e^*}}} f_u$, then the maximum cycle found so far does not define a cover and we apply a procedure that repeatedly tries to add paths to the cycle so as to obtain a violated cycle with ears and tails inequality. This routine is described next.

In each iteration we have a cycle with ears C such that $\sum_{e \in E_C} y'_e < 2$ (in the first iteration it is only a cycle). If $\sum_{i \in V_C} f_i > F$, the cycle with ears C induces a violated inequality. Otherwise, we choose a node i in V_C to be the source and look for a shortest path P between i and some node in $V_C \setminus \{i\}$ in the graph $G - E_C$, where the costs on the edges are the values of the current LP solution y' (see Fig. 5). If such a path exists, we check if $\sum_{e \in E_C \cup E_P} y'_e < 2$ and, if so, we add this ear to C. Moreover, if i and j are the endnodes of the path P we have just added and $(i,j) \in E$, then the coefficient of edge (i,j) is reduced by one (see also Proposition 3.3). If no such path P exists, we try to enlarge the current support (V_C, E_C) by appending tails. Let $\pi y \geqslant 2$ be the inequality generated so far and suppose, $\sum_{i \in V_C} f_i \leqslant F$. We determine a node $i \in V \setminus V_C$ such that for some $j \in V_C$, $y'_{(i,j)} = \min\{y'_{(u,v)}: u \in V \setminus V_C, v \in V_C\}$. If $\pi y' + 2y'_{(i,j)} < 2$ then the edge is added to C and we set $\pi_{(i,j)} = 2$. This gives rise to a new cycle with ears and tails inequality and we continue the above steps until the nodes of the support define a cover.

If the Cycles 1 or Cycles 2 routine returns a cycle (C, E_C) whose nodes define a cover, we try to strengthen the corresponding inequality in the following way. We verify if there exists some chord a in G such that one of the two cycles formed by adding this chord $(C', E_{C'})$ say, defines a cover. If this situation occurs, we can delete the edges in $E_C \setminus E_{C'}$ and the resulting inequality defines a face of higher dimension.

Finally, the routines Cycles 1 and Cycle 2 can be easily adapted to separate cycle inequalities whose node set forms a q-cover. Specifically we iterate until a q-cover is found (instead of a 1-cover), and in the violation test we replace 2 by $q + 1$.

4.3. PBCs

The separation problem for the PBC inequalities is also believed to be NP-hard. Thus we have implemented a heuristic separation routine which looks for violated

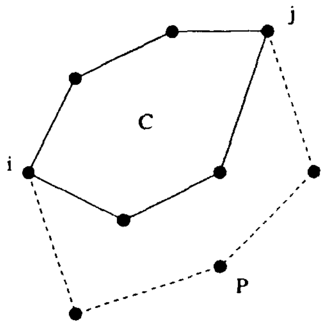

Fig. 5. A cycle with one ear.

PBC inequalities. The support graph of the generated inequalities are PBCs made of two cycles ($r = 2$ in Proposition 3.4), i.e., they are of the form $y(C_1) + y(C_2) \geqslant 4$. Before explaining this routine in more detail, we need to introduce some notation. Let $X = \{x_1, \dots, x_t\}$ be a set of nodes with $t \geqslant 2$ and let V_{C_1} and V_{C_2} be node sets inducing cycles $(V_{C_j}, E_{C_j})(j = 1, 2)$, where $V_{C_1} \cap V_{C_2} = X$. Thus each cycle (V_{C_j}, E_{C_j}) $(j = 1, 2)$ decomposes into t subpaths $\{P_{ij}\}_{i=1}^{t}$ starting at x_i, followed by a possibly empty node set Q_{ij}, and terminating at $x_{(i+1)(\mathrm{mod}\, t)}$. The weight of a node set Q_{ij} is the sum of the weights of the nodes in Q_{ij}. If $S := V_{C_1} \cup V_{C_2}$ and $E_S := E_{C_1} \cup E_{C_2}$, the graph (S, E_S) is called a path-block cycle. Finally, we denote, for all $i = 1, \dots, t$, by $B(i)$ the set of paths $\{P_{ij}\}_{j=1}^{2}$. $B(i)$ is called the ith *block* of the PBC.

The main obstacle in separating PBC inequalities is to satisfy the validity conditions. Essentially, for all the inequalities discussed so far, validity is easily checked since it reduces to verifying that the nodes in the support graph form a cover (possibly a 2-cover). For a PBC (S, E_S) on two cycles, the validity condition implies that, for every block $B(k)$ of the PBC, the nodes in $S \setminus Q_{1k}$ and $S \setminus Q_{2k}$ form covers.

The idea of the separation routine is to enlarge the set of nodes in the support graph while keeping the sum of the node weights in the sets Q_{ij} small. First a node r, called the *root*, is taken such that the degree of r is at least 3 (the routine runs once for every possible choice of such a root node). Then, using the routine *Cycles* 2 (see Section 4.2), a cycle (V_{C_1}, E_{C_1}) is built such that $y'(E_{C_1}) < 2 - \epsilon$ (the value of ϵ used in our implementation is 0.20) and r is in V_{C_1}. The cycle (V_{C_2}, E_{C_2}) initially coincides with (V_{C_1}, E_{C_1}), i.e., the starting PBC has two identical cycles and no blocks. Typically, at this point, the corresponding inequality, given by $y(E_{C_1}) + y(E_{C_2}) = 2y(E_{C_1}) \geqslant 4$ is not valid as the node set $S = V_{C_1} \cup V_{C_2}$ is not a cover (if it is, the routine stops with a violated cycle inequality).

To meet the validity conditions, the PBC is modified so that a new block is created at each iteration. The procedure that creates new blocks is quite involved and we refer to [20] for a detailed description of it. The basic idea is to keep cycle (V_{C_1}, E_{C_1}) unchanged while cycle (V_{C_2}, E_{C_2}) is modified by deleting a subpath of edges that it has in common with E_{C_1} and replacing it by an alternative subpath whose edges are not in E_{C_1} and whose nodes are not in the current set S. At each iteration the current inequality must satisfy $y'(E_{C_1}) + y'(E_{C_2}) < 4$ and, in some sense, we have to ensure that we are getting closer to satisfying the validity condition. This is checked as follows.

Let $\overline{Q}$ be the set Q_{ij} in the current PBC with largest sum of the node weights. The validity of the corresponding inequality implies that $\sum_{i \in S} f_i - \sum_{i \in \overline{Q}} f_i > F$. Thus, new blocks are created in such a way that the value of $\sum_{i \in S} f_i - \sum_{i \in \overline{Q}} f_i$ increases monotonically.

The separation routine stops creating new blocks when one of the following two conditions holds: (i) no new block can be created such that $y'(C_1) + y'(C_2) < 4$, in which case it fails to generate a violated PBC inequality or (ii) the validity condition is satisfied and the current inequality is violated. In the latter case, we check for a possible strengthening of the inequality.

We have implemented two types of strengthenings. For both strengthenings, we check whether there exists a path P in the PBC with nodes $\{v_0, \ldots, v_i, v_{i+1}\}$ such that $(v_j, v_{j+1}) \in E_{C_1} \cap E_{C_2}$, for all $j = 0, \ldots, i$, and $(v_0, v_{i+1}) \in E$. If $\sum_{j\in S} f_j - \sum_{j\in \bar{Q}} f_j - \sum_{j\in\{v_1,\ldots,v_i\}} f_j > F$, then the subpath $v_1, \ldots, v_i$ can be removed and the resulting PBC inequality is still valid. We strengthen the inequality as follows. The coefficients of the edges (v_j, v_{j+1}), for all $j = 0, \ldots, i$, are reduced from 2 to 0 (these edges are removed from E_{C_1} and E_{C_2}) and the coefficient of edge (v_0, v_{i+1}) is increased from 0 to 2 (this edge is added to E_{C_1} and E_{C_2}). It can be easily verified that all points satisfying the original inequality at equality also lie on the face defined by the resulting inequality. This situation is illustrated in Fig. 6 where $|S| = 20$ and we suppose that $F = 16$ and $f_i = 1$ for all $i \in V$.

In the second type of strengthening, we also assume the existence of a path P with nodes $v_0, \ldots, v_{i+1}$ as before such that (v_0, v_{i+1}) is in E. Now suppose that $\sum_{j\in S} f_j - \sum_{j\in \bar{Q}} f_j - \sum_{j\in\{v_1,\ldots,v_i\}} f_j \leqslant F$ and $\sum_{j\in\{v_1,\ldots,v_i\}} f_j < \sum_{j\in S} f_j - F$. In this case a new block can be created by removing the edges (v_j, v_{j+1}), for all $j = 0, \ldots, i$, from C_2 and adding the edge (v_0, v_{i+1}) to C_2. The previous operations correspond to reducing the coefficients of the edges removed from E_{C_2} from 2 to 1 and to increasing the coefficient of edge (v_0, v_{i+1}) from 0 to 1. Again, it is easy to check that the new inequality defines a face of higher dimension in $P^{K,F}(G)$ that contains the face defined by the original inequality. An example of such a situation is illustrated in Fig. 7 where $|S| = 20$ and we suppose that $F = 16$ and $f_i = 1$ for all $i \in V$.

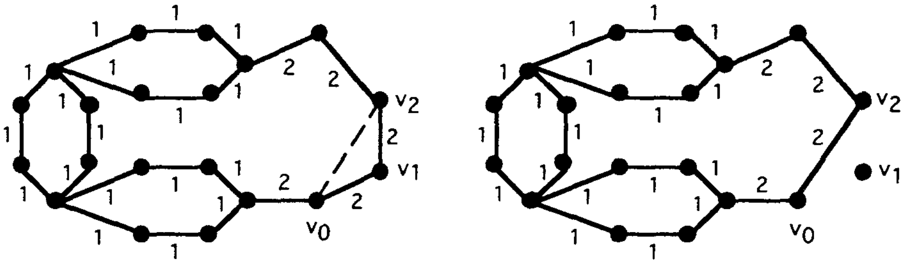

Fig. 6. PBC inequality (strengthening type 1).

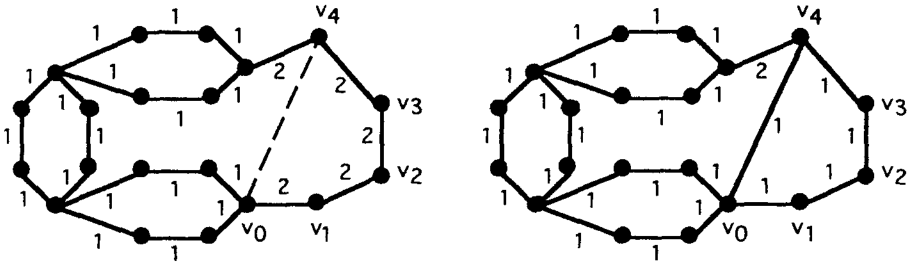

Fig. 7. PBC inequality (strengthening type 2).

4.4. Knapsack trees

Given a graph $G = (V,E)$, we need to find a subtree (T,E_T) of G rooted at a node $r \in V$ and a knapsack tree inequality $\sum_{i\in T} a_i(1 - \sum_{e\in P_i} y_e) \leqslant a_0$. So the first step is to find an appropriate subtree.

Note that, when y is a nonnegative vector, if we delete from (T,E_T) the nodes for which $(1 - \sum_{e\in P_i} y_e) \leqslant 0$, the resulting tree is the support graph of a new knapsack tree inequality which is valid and has a bigger left hand side than the original inequality. Therefore, when looking for violated knapsack tree inequalities, we can concentrate on subtrees of G such that the distance (measured by the y' variables) from the root to any leaf is less than one. Thus, starting from r, the first step in the heuristic is to compute the shortest path tree (T,E_T) with edge lengths y' where $\sum_{e\in P_i} y'_e < 1$ for all $i \in T$.

The second step, having fixed a tree (T,E_T) with root r, is to look for a violated inequality. The weaker form of the inequality: $\sum_{i\in T} a_i(1 - \sum_{e\in P_i} y_e) \leqslant a_0$ where $ax \leqslant a_0$ is a valid inequality for $X = \{x \in \{0,1\}^V : \sum_{i\in T} f_i x_i \leqslant F\}$ with $a \geqslant 0$ can be rewritten as: $\sum_{i\in T} \alpha_i y_{p(i),i} \geqslant \alpha_0$. This immediately suggests two possibilities:

(i) Separate x', where $x'_i = 1 - \sum_{e\in P_i} y'_e$, over the knapsack polytope X. If an inequality $ax \leqslant a_0$ is found, take the possibly strengthened version $\sum_{i\in T} \min\{\alpha_i, \alpha_0, a_0\} y_{p(i),i} \geqslant \alpha_0$ and test for violation of y'.

(ii) $\sum_{i\in T} f_i x_i \leqslant F$ is a valid inequality for X, and thus y' must satisfy the strengthened form: $\sum_{i\in T} \min\{\alpha_i(f), \alpha_0(f), F\} y_{p(i),i} \geqslant \alpha_0(f)$, where $\alpha_i(f), \alpha_0(f)$ are calculated as above from the inequality $\sum_{i\in T} f_i x_i \leqslant F$ in place of $\sum_{i\in T} a_i x_i \leqslant a_0$. Thus we separate y' over the knapsack polytope

$$X' = \{y \in \{0,1\}^E : \sum_{i\in T} \min\{\alpha_i(f), \alpha_0(f), F\} y_{p(i),i} \geqslant \alpha_0(f)\}.$$

Our implementation is thus, with minor modifications, to call two knapsack separation routines over the polyhedra X and X'. The example below demonstrates the two forms of inequality as well as some possibilities for strengthening.

Consider the tree (T,E_T) shown in Fig. 8 where $F = 10$ and $f_i = 1$ for all $i \in T$. Taking $a_i = f_i = 1$ and $a_0 = F = 10$, we obtain the knapsack tree inequality

$$\sum_{i\in T}\left(1 - \sum_{e\in P_i} y_e\right) \leqslant 10$$

or equivalently

$$\begin{aligned} &11y_{1,2} + y_{2,3} + 8y_{2,4} + y_{2,5} + 4y_{4,6} + 3y_{4,7} + 3y_{6,10} + y_{10,11} \\ &\quad + y_{10,12} + y_{7,8} + y_{7,9} + 5y_{1,13} + 4y_{13,14} + 3y_{14,15} + 2y_{15,16} + y_{16,17} \geqslant 7 \end{aligned}$$

which leads to the strengthened inequality

$$\begin{aligned} &7y_{1,2} + y_{2,3} + 7y_{2,4} + y_{2,5} + 4y_{4,6} + 3y_{4,7} + 3y_{6,10} + y_{10,11} \\ &\quad + y_{10,12} + y_{7,8} + y_{7,9} + 5y_{1,13} + 4y_{13,14} + 3y_{14,15} + 2y_{15,16} + y_{16,17} \geqslant 7. \end{aligned}$$

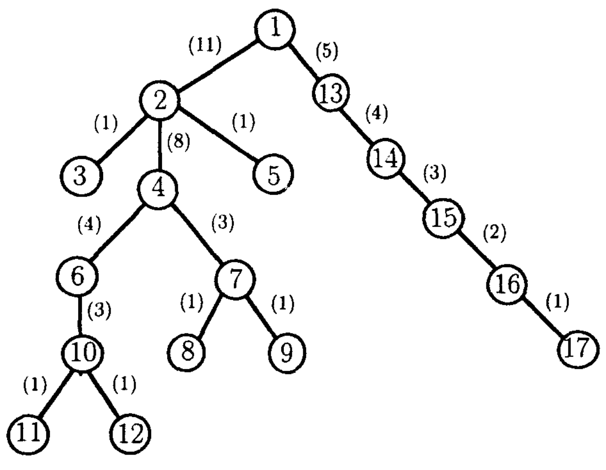

Fig. 8. Strengthening of the knapsack tree inequality.

However, the coefficient of edge (1,2) can be reduced further in the following way. If edge (1,2) is removed from T, we obtain two subtrees (T^1, E_T^1) and (T^2, E_T^2) where $T^1 = \{1, 13, \ldots, 17\}$ and $T^2 = \{2, \ldots, 12\}$. The knapsack tree inequalities corresponding to T^1 and T^2 are given by

$$5y_{1,13} + 4y_{13,14} + 3y_{14,15} + 2y_{15,16} + y_{16,17} \geqslant -4 \quad \text{for } T^1,$$

$$y_{2,3} + 8y_{2,4} + y_{2,5} + 4y_{4,6} + 3y_{4,7} + 3y_{6,10} + y_{10,11} + y_{10,12} + y_{7,8} + y_{7,9} \geqslant 1 \quad \text{for } T^2.$$

Validity is not affected for the inequality of T^1 when the right hand side is set to zero. From these two inequalities we conclude that if edge (1,2) is in a feasible multicut, the second tree also has to be cut at least once, and therefore the coefficient of edge (1,2) can be reduced by one more unit, i.e., it can be reduced from 7 to 6.

In general, given an edge $(p(u), u)$ in T, let (T^u, E_T^u) be the subtree of T hanging from node u and $(T^{p(u)}, E_T^{p(u)})$ be the subtree obtained from T by deleting the nodes in T^u (including u). We then compute the right hand side of the knapsack tree inequalities defined for T^u (rooted at node u) and $T^{p(u)}$ (rooted at r), say α_0^u and $\alpha_0^{p(u)}$, respectively. If the edge $(p(u), u)$ is cut, then $max\{0, \alpha_0^u\}$ must be cut from the tree (T^u, E_T^u) and $\max\{0, \alpha_0^{p(u)}\}$ must be cut from the tree $(T^{p(u)}, E_T^{p(u)})$. Therefore, the coefficient of the edge $(p(u), u)$ can be set to the value

$$\min\{\alpha(u, v), \alpha_0 - \max\{0, \alpha_0^u\} - \max\{0, \alpha_0^{p(u)}\}\}.$$

Note that in the first approach of separating over X, we ignore important information concerning the order relations between the variables, namely the constraints $x_{p(j)} \geqslant x_j$, where $p(j)$ is the tree predecessor of j. Separating over X' partially takes this information into account. Surprisingly perhaps it appears in practice that the choice of the subtree (T, E_T) is at least as important as the choice of the knapsack set.

5. Implementation

There are several features of a branch-and-cut algorithm that require careful tuning to the classes of problem instances. Besides separation algorithms and strategies, there are several other issues that play an important role in influencing the performance of the overall procedure. The general outline of a branch-and-cut algorithm and its basic issues can be found in [1] or [18] among others.

5.1. Initial LP

In our implementation, the first LP consists of the equations $\sum_{k=1}^{K} z_i^k = 1$ for all $i \in V$ and the knapsack inequalities $\sum_{i\in V} f_i z_i^k \leqslant F$ for all $k \in M := \{1,\dots,K\}$. Note that these two types of inequalities do not give an integer programming formulation for the node capacitated graph partitioning problem, because the edge constraints $\sum_{k\in K_1} z_i^k + \sum_{k\notin K_1} z_j^k - y_{(i,j)} \leqslant 1$ do not appear in the first LP. The reason for this choice is that, for every edge $(i,j) \in E$, there exist exponentially many edge constraints and adding them all explicitly to the LP is impossible. Moreover, we can solve the separation problem for these inequalities in polynomial time using the following procedure.

Given $(i,j) \in E$, we compute $z = \max\{z_i^k, z_j^k\}$ for every $k \in M$. If $z = z_i^k$, we add k to set K_1.

If $K_1 = M$, this procedure yields an inequality $\sum_{k\in M} z_i^k - y_{(i,j)} \leqslant 1$ which cannot be violated because of the equation $\sum_{k\in M} z_i^k = 1$. Otherwise $K_1 \neq M$, and it is easy to see that the edge constraint corresponding to K_1 is the one with maximum slack for edge (i,j).

Based on these observations, our approach is to add the edge constraints "on the fly", as they are needed.

5.2. LPs

As LP solver we use CPLEX3.0. In order to control the size of the LPs, we restrict the number of cutting planes to be added in one step by some parameter p. If the number of violated inequalities found by our separation routines exceeds this parameter, we use the following strategy to select p inequalities.

Let s be a scaling parameter (typically, $s = 20$). We generate a "double hash table" by using the hash function $h(\text{inequality}) := (s \times \text{slack}, \text{type})$. Here "slack" denotes the violation of the inequality and "type" its type, i.e., tree, cycle, etc. We select p (typically, $p = 1000$) inequalities according to decreasing values of $s \times$ slack. Among the inequalities with the same value of $s \times$ slack, we select inequalities of different types by exhausting the buckets in cyclic order.

Since different separation algorithms may generate the same inequality, we test for redundancy of the inequalities to be selected. As a byproduct of our hash implementation we only need to check redundancy among the inequalities with the same hash function value. Inequalities in the LP and the inequalities with different hash function values are guaranteed to be different.

Another way to keep the size of the LPs under control is to eliminate inequalities with large slack. These inequalities are added to a "pool" and kept for a certain number of iterations. In each iteration of the branch-and-cut algorithm we scan through the pool and look for violated inequalities. If such inequalities are found, they are inserted into the hash data structure for violated inequalities and removed from the pool. The inequalities that are not violated for a certain number of iterations are removed from the pool by a garbage disposer. Finally, we have implemented a procedure for fixing variables based on reduced cost arguments and on logical implications.

5.3. Branching strategy

For the test problems introduced in the next section, we have chosen a fixed strategy. We add cutting planes given by our separation algorithms until no more violated inequalities are found or *tailing off* is observed, i.e., the improvement of the lower bound over several iterations is below a given threshold parameter (in the final implementation 1% improvement over 10 iterations). We then proceed to the enumerative phase of our code.

To branch, we choose a variable whose value is closest to 0.5, and we create two subproblems by fixing this variable to 0 and 1, respectively. Priority is given to branching on the edge variables. Each of the subproblems generated is represented by its branching node and added to the list of active nodes. An active node is selected with the minimum value of the local lower bound associated with the subproblem. The initial LP for this node consists of the constraints of the last LP loaded plus the variable fixing corresponding to the node. The same set of separation routines are then called at each node.

5.4. Primal heuristics

We have implemented two LP-based heuristics that try to find feasible partitions of the graph, and an improvement heuristic. Suppose that y' is the current fractional LP solution.

The first heuristic called the *Edge* heuristic only uses the information provided by the edge variables to generate an upper bound. It is based on the idea that, if y'_{ij} is close to 0, nodes i and j will probably be together in some cluster of the partition. Thus, given a positive (small) value ϵ, let E' be the set of edges in E satisfying $y'_e < \epsilon$. Consider now the problem of finding a minimum cost forest T in (V, E') such that: *(i)* the number of components in T is at most K (the size of a feasible partition) and *(ii)* the sum of the node weights in each component in T is at most F (the cluster capacity). Clearly, any solution of this problem immediately converts into a solution of the graph partitioning problem by making a one-to-one correspondence between the components of the forest and the clusters in the partition.

Therefore, a modified version of Kruskal's algorithm for the minimum spanning tree problem is used to find the forest T. The aim of these modifications is to avoid the creation of components for which the sum of the node weights is larger than F. If this modified version of Kruskal's algorithm terminates with a forest that has at most K components, a feasible solution for the graph partitioning problem is available. Otherwise, the number of components in T exceeds K and the *Edge* heuristic tries to find a feasible solution for the bin packing problem in which the components in T are interpreted as items that have to be packed into bins (cluster) of size F. If a solution is found for the bin packing problem with cost at most K, then a feasible solution for the partitioning problem is available. Otherwise, the heuristic fails to produce a feasible partition of G and is unable to give an upper bound.

The second heuristic called the *Random* heuristic, uses the node variable information. It is based upon a heuristic for the Unconstrained Global Routing Problem in VLSI-design (see [17]). Since we have the equality constraints $\sum_{k=1}^{K} z_u^k = 1$ in our model, we can interpret the z_u^k variables as a probability distribution for node u. For instance, for an equipartition problem, suppose that we have a fractional solution such that $z_u^1 = 0.6$ and $z_u^2 = 0.4$. In this case, we assume that node u has 60% probability of being in cluster 1 and 40% of being in cluster 2.

The heuristic simulates a series of experiments in which the nodes are assigned to the clusters according to these probability distributions and, among the solutions generated, the feasible solution of minimum cost is chosen. At the beginning of an experiment, the n nodes of the graph are unassigned, and the experiment consists of n iterations. At each iteration an unassigned node u is chosen and it is assigned to cluster k with probability z_u^k. If the capacity remaining in cluster k is less than w_u, then a new trial has to be made for node u since this assignment is infeasible. In the new trial, the probability distribution is adjusted so as to prevent node u from reassignment to cluster k. This is repeated until either there are no more clusters available in which u fits and the experiment is aborted, or a cluster is found that can accept node u and the iteration terminates. The number of times the heuristic is called is proportional to the number of fractional variables.

Besides these primal heuristics, we have implemented an *Improvement* heuristic. This routine is a local search procedure based on ideas of Fiduccia and Mattheyses [10]. Iteratively, either two nodes assigned to different clusters are exchanged or some node is assigned to a different cluster. We have evaluated the performance of the primal heuristics and the Improvement heuristic in a series of tests. The best trade-off between running time and quality of the solution has been reached by a combination of the Edge and Improvement heuristics.

6. Computational results

We have applied our branch-and-cut algorithm to test instances arising from the applications discussed in Section 2. We now briefly report on the strategy chosen to attack these instances and then present the computational results.

Having designed and implemented various separation algorithms for different classes of inequalities, it is necessary to decide for each class of instances which separation routines to call and in what order. To answer these questions we performed a series of tests. We always tried each separation routine individually. We have also chosen many sequences of separation routines (Cycles 1, Cycles 2, the separation routine for knapsack trees, for instance) and called this sequence of algorithms after every solution of the current linear programming relaxation. We examined these differing sequences both for the running time and the quality of the upper and lower bounds. Here we restrict ourselves to a discussion of the conclusions drawn and a presentation of the final strategies chosen.

1. For all the test instances, cycle and knapsack tree inequalities are indispensable for obtaining satisfactory lower bounds.
2. Tree and star inequalities are useful only for the compiler design instances. One possible explanation is that the corresponding graphs contain a small number of nodes with very high node weight and degree and that star inequalities are well suited to these graphs.
3. The PBC inequalities are strong for the mesh application problems. More precisely, for all of the smaller examples, the number of PBC inequalities needed in order to prove optimality is significantly less than the number of cycle (with ears and tails) and knapsack tree inequalities needed. However, the PBC separation routine is very expensive in terms of running time, and it is not used in our final runs.
4. We have also compared the two cycle routines Cycles 1 and Cycles 2 for the q-cover cycle inequalities, where in our implementation we only consider the cases $q = 1$ and $q = 2$. Though we have not drawn any final conclusion, we observe that (a) cycle inequalities are rarely sufficient to solve any of the examples to optimality and in some cases the lower bound is still more than 50% away from the optimum solution and (b) the Cycles 1 routine usually provides better lower bounds than does the Cycles 2 routine, at the price of a higher running time.
5. It is cheap (with respect to the running time) to do the strengthening of the inequalities discussed in Section 4. Optimal solutions are reached faster using the strengthened inequalities.
6. In spite of using strengthened inequalities, the number of inequalities generated, especially cycle and knapsack tree inequalities is high, often several thousand inequalities are generated in all.

For the final runs we call the Cycles 2 routine for the mesh and VLSI examples, and we call both routines for the compiler design instances. For the problem instances we use the following convention. The compiler design instances start with *cb*, the instances associated with the mesh application start with *mesh* and the VLSI instances start with *vlsi*. The first number corresponds to the number of nodes in the graph, the second number is the upper bound on the number of elements of the partition and the third number reflects the number of edges in the graph. Thus, "vlsi.42.4.132" is a VLSI instance that involves 42 nodes and 132 edges, and the task is to partition the node set into no more than four elements.

Table 1
Computational results for compiler design instances

Problem	lb	ub	BC	Total time	Root lb	Time
cb450.30.30.47	827	827	1	26	827	26
cb450.30.30.56	1217	1217	1	16	1217	16
cb450.45.45.98	3320	3320	9	20:44	3304	11:28
cb450.47.47.99	2069	2069	9	18:43	2060	7:31
cb450.47.47.101	3585	3585	119	163:25	3518	6:35
cb450.61.61.187	10 912	11 064	7	⋆ 300:00	10 805	113:24
cb512.30.30.47	752	752	1	21	752	21
cb512.30.30.56	1111	1111	1	34	1111	34
cb512.45.45.98	3010	3010	5	17:04	3000	12:39
cb512.47.47.99	1913	1913	15	56:50	1903	11:32
cb512.47.47.101	3194	3194	9	26:09	3147	10:50
cb512.61.61.187	9716	9716	3	176:47	9709	173:03

All results were obtained on a Sun 4/50 with CPLEX version 2.2 used as the linear programming solver.

6.1. Compiler design problems

These problems have been introduced by Johnson et al. [14] who solved 9 out of 12 [1] problem instances to optimality using column generation techniques. Note that their objective is to maximize the sum of weights of edges within clusters while ours is to minimize the sum of the weights of the edges in the multicut. We have run our branch-and-cut algorithm on these instances by calling the separation routines for trees, stars and knapsack trees as well as the routines Cycles 1, Cycles 2. To obtain an upper bound, we apply the Edge heuristic with the Improvement heuristic. The results are shown in Table 1. In the second and third columns the final lower and upper bound, respectively, are given after spending at most 300 min of CPU time. The number of branch-and-cut nodes explored during the run is given in column 4. The CPU times measured in minutes and seconds on a SUN 4/50 are presented in column 5. The symbol ⋆ in column 5 means that we could not solve the corresponding example to optimality. Finally, columns 6 and 7 contain the values of the lower bound in the root node of the branch-and-cut tree and the CPU time that is needed to achieve the corresponding lower bound.

As shown in Table 1 we have solved all problems to optimality except the instance cb450.61.61.187. For this instance the results after several more hours of computation indicate that days of computation time would be necessary to prove optimality.

[1] It is mistakenly claimed in [14] that cb512.47.47.101 has no duality gap. Thus 9 rather than 10 instances have no gap.

A comparison of our results to those presented in [14] is difficult, since the codes are implemented on different machines and different LP solvers are used. However, we prove optimality for two of the three instances not solved to optimality there.

In Table 6 in Appendix A, we report on the number of inequalities generated of each type for the compiler design, mesh and VLSI instances.

6.2. Mesh problems

The instances for this application are based on finite element meshes appearing in [20,21]. Many of these practical instances give rise to graphs with hundreds or thousands of nodes which are too large for our approach (even solving an initial LP may be impossible). However, there are some smaller meshes that we could tackle. In particular, instance mesh.274.2.469 is taken from [21] and instance mesh.148.2.265 is taken from [19]. Instances mesh.138.2.232, mesh.70.2.120 and mesh.31.2.50 correspond to meshes we have generated by considering parts of some other larger meshes. These parts of meshes have been obtained by a reasonable splitting criterion based upon certain symmetries of the meshes.

Our exact branch-and-cut algorithm was applied to the instances described above. The data set consists of five equipartition instances with between 31 and 274 nodes and between 50 and 469 edges. Thus $K = 2, f_i = 1$ for all $i \in V$ and $F = \lceil n/2 \rceil$. In our runs we call the Cycles 2 and knapsack tree separation routines and a combination of the Edge and the Improvement heuristic to find upper bounds. The results obtained are shown in Table 2. We are able to solve all these equipartition problems at the root node of the branch-and-cut tree.

In many mesh applications the number of clusters is $K = 2^p$. A typical approach to solve such problems heuristically, called the *hierarchical approach*, is to solve a series of $2^p - 1$ equipartition problems. Using our code and taking $p = 2$, we have compared the heuristic solution obtained by solving a series of three equipartition problems to optimality against the optimal solution obtained by directly partitioning the mesh into $K = 4$ parts directly. These results are shown in Table 3.

For the test instances the strategy of solving a series of equipartition problems is apparently superior. For the example mesh.70.120 the hierarchical approach does

Table 2
Computational results for mesh problems

Problem	lb	ub	BC	Total time	Root lb	Time
mesh.31.2.50	6	6	1	5	6	5
mesh.70.2.120	7	7	1	1:43	7	1:16
mesh.138.2.232	8	8	1	108:36	8	108:36
mesh.148.2.265	7	7	1	14:11	7	7:11
mesh.274.2.469	7	7	1	92:02	7	92:02

Table 3
Splitting tests for mesh problems

Problem	Partition into 4		Series of equipartitions	
	Optimum value	CPU time	Optimum value	Total time
mesh.31.2/4.50	12	34	12	8
mesh.70.2/4.120	14	2:38	16	1:52
mesh.148.2/4.265	23	2313:15	23	22:40

not produce the optimum solution for the partition into four parts, but in the other two problems the solutions obtained by both methods have identical values. The total CPU time used for solving the three equipartition problems is far less than the time needed for solving the 4-partitioning problems directly.

This suggests that good solutions for the mesh partitioning problem can perhaps be obtained with a hierarchical approach, provided that the equipartitioning problems are solved to optimality.

6.3. *VLSI problems*

Typical instances arising in the layout of electronic circuits involve several thousands of cells [15,23] and even when trying to split a circuit with 1000 cells into four pieces, it takes several hours to solve the first linear programming relaxation. In order to create instances of smaller size, we have executed one iteration of the placement and decomposition code developed in [23,15] in which the cells are distributed on 16 locations on the master. All the cells that are assigned to the same position of the master form a subcircuit suitable for partitioning into smaller pieces. The instances we obtained in this way involve between 17 and 278 nodes and between 39 and 504 edges.

Our separation strategy is to use the Cycles 2 and knapsack tree separation routines. The upper bounds are computed by the Edge and Improvement heuristics. In Table 4 we summarize the results. For the instances up to 50 nodes, the gap between lower and upper bounds is very small. Indeed, for eight out of nine instances optimality is proved without too much branching. For the instances beyond 150 nodes we have not succeeded in proving optimality and the gap between lower and upper bounds lies between 5% and 30%.

In circuit design a common approach to solving partitioning problems heuristically is again to proceed hierarchically, i.e., instead of partitioning the graph (circuit) into $K = 2^p$ parts, a series of $2^p - 1$ equipartition problems is solved. We have compared both strategies on small examples with $p = 2$. The results are shown in Table 5.

Contrary to the results for the mesh problems, the hierarchical strategy does not appear to be competitive with the direct approach. In all four examples, the solution

Table 4
Computational results for VLSI problems

Problem	lb	ub	BC	Total time	Root lb	Time
vlsi.17.4.39	44	44	27	17:07	41	1:07
vlsi.15.4.29	55	55	1	8	55	0:08
vlsi.34.4.71	78	78	3	1:33	77	1:30
vlsi.37.4.92	156	156	1	1:39	156	1:39
vlsi.38.4.105	275	277	75	⋆ 300:00	272	15:11
vlsi.42.4.132	256	256	3	20:10	255	19:29
vlsi.44.4.105	164	164	5	2:00	163	1:39
vlsi.46.4.79	94	94	35	59:15	90	6:22
vlsi.48.4.81	99	99	267	300:00	94	7:36
vlsi.166.4.504	179	244	1	⋆ 300:00	179	300:00
vlsi.239.4.375	281	297	7	⋆ 300:00	281	200:00
vlsi.278.4.396	358	437	1	⋆ 300:00	358	300:00

Table 5
Splitting tests for VLSI problems

Problem	Partition into 4		Series of equipartitions	
	Optimum value	CPU time	Optimum value	CPU time
vlsi.17.2/4.39	44	17:07	46	3:06
vlsi.15.2/4.29	55	8	58	5
vlsi.37.2/4.92	156	1:39	163	20:40
vlsi.42.2/4.132	256	20:10	314	12:17

found by the hierarchical strategy is not an optimum solution of the original problem. Moreover it appears that, the larger the graph, the larger the gap between the heuristic and the optimum solution. For one instance the time needed to solve the problem optimally is even smaller than the time needed to obtain the heuristic solution. These results suggest that it may not be worthwhile spending too much effort in solving equipartitioning problems to optimality when using the hierarchical approach to circuit partitioning problems.

7. Conclusions

The results presented above can be interpreted in different ways. They are disappointing in the sense that we are still a long way from solving problems of the size that occur in many applications: mesh problems can easily have 10^3–10^6 nodes and VLSI problems are nearly as large. The largest problem we have solved to optimality is a mesh problem with 274 nodes and 469 edges.

On the other hand, our branch-and-cut routine appears to be a step forward. We are not aware of any literature in which this general model has been studied from a polyhedral point of view, or in which such a variety of node capacitated graph partitioning problems have been solved to optimality. Using our code, two more of the compiler design problems of [14] can be solved, equipartitioning problems with more than 200 nodes have been solved, something which initially appeared to be a very distant goal, and finally as far as we can judge the VLSI problems solved are far from trivial.

Comparing our results to those in [6], involving equipartitioning problems on complete graphs, is difficult, because the test instances treated there are dense and, our code handles a much more general class of partitioning problems on sparse graphs. It does however seem likely that, as for special cases like quadratic optimization or max cut, one of the crucial factors in measuring problem difficulty is edge density.

Another point of comparison is the column generation approach applied by Johnson et al. [14] and Vanderbeck [22]. The latter has very recently tackled our test set with an exact integer programming column generation approach. For the majority of the smaller instances his resolution times are of the same order of magnitude as ours. In several cases the bound he obtains at the first node of the branch-and-bound tree is superior to ours, which indicates that there is considerable room for improvement in our cutting plane approach.

Our limited tests comparing 4-partitioning with a hierarchical use of equipartitioning can certainly not be judged conclusive. However, they suggest that a hierarchical use of equipartitioning may be effective for mesh problems. On the other hand, our results on VLSI instances suggest that here one should try working directly on the multipartitioning problem. Such tentative conclusions (though restricted to problems of small size) can only be drawn because the associated instances have been solved to optimality. Verifying whether similar conclusions carry over to problems of bigger size is obviously an open question and a major challenge.

Acknowledgements

This research was supported in part by, Science Program SC1-CT91-620 of the EEC, contract ERB CHBGCT 920167 of the EEC Human Capital and Mobility Program and the contract No. 26 of the "Programme Pôle d'Attraction Interuniversitaire" of the Belgian Government. Cid de Souza was partially supported by a doctoral fellowship from the "Conselho Nacional de Desenvolvimento Científico e Tecnológico" (Brazil).

Appendix A

The number of inequalities generated for the compiler design, mesh and VLSI instances are given in Table 6.

Table 6
Cutting plane statistics with the default strategy

Example	Trees/stars	Cycles	Knapsack trees
cb450.30.30.47	898	294	833
cb450.30.30.56	783	361	533
cb450.45.45.98	11 775	3865	7723
cb450.47.47.99	11 641	3843	9220
cb450.47.47.101	121 325	29 634	77 071
cb450.61.61.187	148 048	29 639	42 185
cb512.30.30.47	855	307	864
cb512.30.30.56	1202	568	1193
cb512.45.45.98	11 748	3509	6925
cb512.47.47.99	24 458	8029	17 642
cb512.47.47.101	21 313	5201	12 283
cb512.61.61.187	132 323	24 158	31 016
mesh.31.2.50	*	60	196
mesh.70.2.120	*	418	1561
mesh.138.2.232	*	4830	23 228
mesh.148.2.265	*	740	2509
mesh.274.2.469	*	1918	6207
vlsi.17.4.39	*	2923	35 879
vlsi.15.4.29	*	52	1056
vlsi.34.4.71	*	741	4444
vlsi.37.4.92	*	815	4723
vlsi.38.4.105	*	49 795	235 539
vlsi.42.4.132	*	4820	18682
vlsi.44.4.105	*	1001	3797
vlsi.46.4.79	*	3249	85572
vlsi.48.4.81	*	23 154	411 911
vlsi.166.4.504	*	11 770	71 756
vlsi.239.4.375	*	0	44 514

Columns 2–4 represent the number of inequalities generated in solving the given instance; the $*$ in column 2 for the rows corresponding to the mesh and vlsi instances indicate that the separation routines for stars and trees were not called.

References

[1] D. Applegate, R.E. Bixby, V. Chvátal, W. Cook, Finding cuts in the TSP, DIMACS Technical Report, 1994, pp. 95–05.

[2] F. Barahona, A. Casari, On the magnetisation of the ground states in two dimensional Ising spin glasses, Computer Physics Communications 49 (1988) 417–421.

[3] F. Barahona, M. Grötschel, M. Jünger, G. Reinelt, An application of combinatorial optimization to statistical physics and circuit layout design, Operations Research 36 (1988) 493–513.

[4] F. Barahona, A. Mahjoub, On the cut polytope, Mathematical Programming 36 (1986) 157–173.

[5] N. Boissin, Optimisation des Fonctions Quadratiques en Variables Bivalentes, Thèse de Doctorat, Conservatoire National des Arts et Métiers, Paris, 1994.

[6] L. Brunetta, M. Conforti, G. Rinaldi, A branch-and-cut algorithm for the resolution of the equicut problem, Working Paper no. 361, IASI-CNR, Rome, 1993.

[7] S. Chopra, M.R. Rao, On the multiway cut polyhedron, Networks 21 (1991) 51–89.

[8] C.E. Ferreira, A. Martin, C.C. de Souza, R. Weismantel, L.A. Wolsey, The node capacitated graph partitioning problems: formulations and valid inequalities, Mathematical Programming 74 (1996) 247–267.

[9] C.E. Ferreira, A. Martin, R. Weismantel, A cutting plane based algorithm for the multiple knapsack problem, SIAM J. on Optimization 6 (1996) 858–877.

[10] C.M. Fiduccia, R.M. Mattheyses, A linear time heuristic for improving network partitionings, in: Proceedings of the 19th Design Automation Conference, Las Vegas, 1982, pp. 175–181.

[11] M. Grötschel, Y. Wakbayashi, A cutting plane algorithm for a clustering problem, Mathematical Programming Series B 45 (1989) 59–96.

[12] S. Holm, M.M. Sorensen, The optimal graph partitioning problem, OR Spektrum 15 (1993) 1–8.

[13] D.S. Johnson, C.R. Aragon, L.A. McGeoch, C. Schevon, Optimization by simulated annealing: an experimental evaluation: Part I, Graph partitioning, Operations Research 37 (1989) 865–892.

[14] E. Johnson, A. Mehrotra, G.L. Nemhauser, Min-cut clustering, Mathematical Programming 62 (1993) 133–152.

[15] M. Jünger, A. Martin, G. Reinelt, R. Weismantel, Quadratic 0/1 optimization and a decomposition approach for the placement of electronic circuits, Mathematical Programming 63 (1994) 257–279.

[16] W. Kernighan, S. Lin, An efficient heuristic procedure for partitioning graphs, Bell Systems Technical Journal 49 (2) (1970) 291–307.

[17] T. Lengauer, Combinatorial Algorithms for Integrated Circuit Layout, Wiley, New York, 1990.

[18] M.W. Padberg, G. Rinaldi, A branch and cut algorithm for the resolution of large-scale symmetric traveling salesman problems, SIAM Review 33 (1991) 60–100.

[19] H.L.G. Pina, An algorithm for frontwidth reduction, International Journal on Numerical Methods in Engineering 17 (1981) 1539–1546.

[20] C.C. de Souza, The graph equipartition problem: optimal solutions, extensions and applications, Doctoral Thesis, Faculté des Sciences Appliquées, Université Catholique de Louvain, Louvain-la-Neuve, Belgium, 1993.

[21] C.C. de Souza, R. Keunings, L.A. Wolsey, O. Zone, A new approach to minimising the frontwidth in finite element calculations, Computer Methods in Applied Mechanics and Engineering 111 (1994) 323–334.

[22] F. Vanderbeck, Decomposition and column generation for integer programs, Doctoral Thesis, Faculté des Sciences Appliquées, Université Catholique de Louvain, Louvain-la-Neuve, Belgium, 1994.

[23] R. Weismantel, Plazieren von Zellen: Theorie and Lösung eines quadratischen 0-1 Optimierungsproblem, Dissertation, Technische Universität, Berlin, 1992 .