ISTITUTO DI ANALISI DEI SISTEMI ED INFORMATICA
CONSIGLIO NAZIONALE DELLE RICERCHE

L. Brunetta, M. Conforti, G. Rinaldi

A BRANCH-AND-CUT ALGORITHM FOR THE EQUICUT PROBLEM

R. 361 Luglio 1993

Lorenzo Brunetta, Michele Conforti – Dipartimento di Matematica Pura e Applicata, Università degli Studi di Padova, via Belzoni 7 - 35131 Padova, Italy..

Giovanni Rinaldi – Istituto di Analisi dei Sistemi ed Informatica del CNR, Viale Manzoni 30 - 00185 Roma, Italy..

ISSN: 1128–3378

Collana dei Rapporti
dell'Istituto di Analisi dei Sistemi ed Informatica, CNR

viale Manzoni 30, 00185 ROMA, Italy

tel. ++39-06-77161
fax ++39-06-7716461
email: iasi@iasi.rm.cnr.it
URL: http://www.iasi.rm.cnr.it

Abstract

We describe an algorithm for solving the equicut problem on complete graphs. The core of the algorithm is a cutting-plane procedure that exploits a subset of the linear inequalities defining the convex hull of the incidence vectors of the edge sets that define an equicut. The cuts are generated by several separation procedures that will be described in the paper. Whenever the cutting-plane procedure does not terminate with an optimal solution the algorithm uses a branch-and-cut strategy. We also describe the implementation of the algorithm and the interface with the LP solver. We then report on our computational results.

Key words: equicut, max-cut, polyhedral theory, cutting-plane algorithm, heuristic algorithm, branch-and-cut.

1. Introduction

We consider an undirected graph $G = (V, E)$, where E is the edge set and V is the vertex set, with edge weights $c_e \in \mathbb{R}$, $e \in E$. We denote by uv the edge of E having u and v as endpoints. For a (possibly empty) subset S of V, a *cut* associated with S is the following subset of E:

$$\delta(S) = \{uv \in E : u \in S,\, v \in V \setminus S\}.$$

The sets S and $V \setminus S$ are called the *shores* of the cut $\delta(S)$. To denote a cut defined by a single node, we simply write $\delta(v)$ instead of $\delta(\{v\})$. A cut is an empty set of edges if $S = \emptyset$ or $S = V$. The weight $c(\delta(S))$ of a cut $\delta(S)$, is the sum of the weights of its edges. A cut $\delta(S)$ is an *equicut* if $|S| = \lfloor \frac{|V|}{2} \rfloor$ or $|S| = \lceil \frac{|V|}{2} \rceil$.

In this paper we describe a branch-and-cut algorithm for finding an equicut of minimum weight in K_{2p}, a complete graph with an even number of nodes. The problem is known to be $\mathcal{NP}$ hard. By possibly adding one node to the graph and connecting it to the other nodes with edges of weight zero, our algorithm solves the minimum weight equicut also if the graph has an odd number of nodes. If the graph is not complete, by adding all the missing edges to the graph and assigning a zero weight to them, one obtains an equivalent minimum weight equicut problem on a complete graph. Finally, since there are no sign restrictions on the weight vector c, searching for a minimum is equivalent to searching for a maximum weight equicut.

The relevance of this problem arises in several areas. In Physics, for example, since it models the problem of finding the ground state magnetization of spin glasses having zero magnetic field. In VLSI design it models the problem of minimizing the number of vias (holes on a printed circuit board, or contacts on a chip) subject to pin preassignment and layer preferences. For these applications of the problem, see Barahona, Grötschel, Jünger and Reinelt [2] for an extensive survey.

The four basic components of a branch-and-cut algorithm are: a heuristic procedure to find a good upper bound on the objective function, a set of exact and heuristic procedures for the separation of violated inequalities belonging to a partial description of the equicut polytope, an interface with the LP solver, and an enumeration procedure that combines branching with cutting-planes techniques. We describe all these components in the following sections.

We give an overview of the algorithm in Section 2. Then we describe the heuristic procedure in Section 3, the constraint generation procedure in Section 4, and the implementation of the branch-and-cut algorithm in Section 5. Finally, we report on our computational experience with the algorithm in Section 6.

2. The branch-and-cut algorithm

A *subgraph* of G is the graph $G' = (S, F)$, where $S \subseteq V$ and $F \subseteq E$. The subgraph $G' = (S, E(S))$ is said to be *induced* by the node set $S \subseteq V$ if $E(S)$ is the set of edges having both end nodes in S. We denote with K_q the subgraph of K_{2p} induced by a set $S \subseteq V$ of q nodes.

2.1. Polyhedral results

We denote by $\mathbb{R}^E$ the real vector space whose components are indexed by the elements of E. With every subset $F \subseteq E$ we associate an *incidence vector* $x^F \in \mathbb{R}^E$ where a component x^F_e is equal to 1 if $e \in F$, and to 0 otherwise. For $y \in \mathbb{R}^E$ and $S \subseteq E$ we indicate with $y(S)$ the sum $\sum_{e \in S} y_e$.

The *cut polytope* $C(G)$ and the *equicut polytope* $Q(G)$ of the graph G are the convex hulls of incidence vectors of all cuts and of all equicuts of G, respectively.

Therefore the minimum weight equicut of G can be found, in principle, by solving the following linear program:

$$\begin{aligned} &\min\ cx \\ &\text{s.t. } x \in Q(G). \end{aligned} \tag{2.1}$$

The optimal solution to (2.1) is the incidence vector of an optimal equicut. Unfortunately, it is unknow how to completely describe $Q(G)$ by linear inequalities. However, using a partial set of the inequalities defining $Q(G)$, which yield a relaxation of $Q(G)$, it is still possible to find an optimal solution to (2.1) using a branch-and-cut algorithm.

Conforti, Rao, and Sassano [3] characterize the p-dimensional affine hull of $Q(K_{2p})$ in the following theorem.

Theorem 2.1. *For $p \geq 2$, the dimension of $Q(K_{2p})$ is $\binom{2p}{2} - 2p$ and the affine hull of $Q(K_{2p})$ is*

$$\{x \in \mathbb{R}^E : A_{2p} x = \mathbf{p}\},$$

where A_{2p} is the $(p(2p-1) \times 2p)$ node-edge incidence matrix of K_{2p} and $\mathbf{p}$ is the vector of $\mathbb{R}^E$ with all the components equal to p.

Throughout the paper we assume that $p \geq 2$.

An edge set $F \subseteq E$ is said to be a *p-matching* if every node in the subgraph $G' = (V, F)$ has degree less or equal to p. A p-matching is a *p-factor* if graph G' is regular of degree p.

Edmonds and Johnson [7] show that the following system of inequalities describes the *p-factor polytope* $PF_p(G)$ of G, i.e., the convex hull of the incidence vectors of all p-factors in a graph G:

$$\begin{aligned} &x(\delta(v)) = p && \text{for all } p \in V \\ &0 \leq x_e \leq 1 && \text{for all } e \in E \\ &x(E(W)) + x(T) \leq \tfrac{1}{2}(p|W| + |T| - 1) && \text{for all } W \subseteq V, T \subseteq \delta(W), \\ & && \quad p|W| + |T| \text{ odd}. \end{aligned} \tag{2.2}$$

The last set of inequalities in (2.2) are called the *matching constraints*.

Equicuts are related to p-factors as observed by Conforti, Rao and Sassano [3] in the following proposition.

Proposition 2.2. *Let F be a p-factor of a complete graph K_{2p}. Then F is an equicut if and only if $F \subseteq \delta(S)$ (or, equivalently, $F = \delta(S)$) for some $S \subseteq V$.*

Let us denote by $\tilde{C}(G)$ an LP relaxation of $C(G)$, i.e., a polytope (containing $C(G)$) whose integer points are the incidence vectors of all the cuts of G. Then Proposition 2.2 implies that the intersection of $\tilde{C}(G)$ and $PF_p(G)$ yields an LP relaxation of $Q(G)$, i.e.,

$$Q(G) = conv\{PF_p(G) \cap \tilde{C}(G) \cap \mathbb{Z}^E\}$$

An LP relaxation of $C(K_{2p})$ is given by the following system of inequalities, called the *triangle inequalities* (see Barahona and Mahjoub [1]):

$$\left.\begin{array}{r} x_{ij} + x_{i\ell} + x_{j\ell} \leq 2 \\ x_{ij} - x_{i\ell} - x_{j\ell} \leq 0 \\ -x_{ij} + x_{i\ell} - x_{j\ell} \leq 0 \\ -x_{ij} - x_{i\ell} + x_{j\ell} \leq 0 \end{array}\right\} \quad \begin{array}{c} \text{for all triples } i,j,\ell \in \{1,...,2p\}, \\ i \neq j \neq \ell \neq i. \end{array} \tag{2.3}$$

Consequently, the inequalities (2.2) and (2.3) define an LP relaxation $\tilde{Q}(K_{2p})$ of $Q(K_{2p})$ and provide an integer programming formulation for the equicut problem on K_{2p}.

The optimal solution $\tilde{x}$ to the problem $\max\{cx : x \in \tilde{Q}(K_{2p})\}$ is not in general an integer vector, thus it does not directly provide an optimal solution to the equicut problem. However, the optimal objective function value $c\tilde{x}$ can be used as a lower bound on the value of a minimal weight equicut in an enumeration algorithm like the branch-and-bound or the branch-and-cut.

The smaller the polytope $\tilde{Q}(K_{2p})$ (and thus the higher the value of the lower bound), the faster is the enumeration algorithm in finding an optimal solution to the equicut problem. Therefore, to produce an LP relaxation that is tighter than $\tilde{Q}(K_{2p})$ we add other inequalities, which belong to two classes, to those defining $\tilde{Q}(K_{2p})$. These new inequalities are described in the following two theorems, where it is claimed that they define facets of $Q(K_{2p})$. For the proofs of these theorems see Conforti, Rao, and Sassano [4].

Theorem 2.3. *For a proper nonempty induced subgraph K_q of K_{2p}, the inequality*

$$x(E(K_q)) \leq \left\lceil \frac{1}{2}q \right\rceil \left\lfloor \frac{1}{2}q \right\rfloor \tag{2.4}$$

defines a facet of $Q(K_{2p})$ if and only if q is odd and at least 3.

Remark. From the previous theorem we have, as a corollary, that the triangle inequality $x_{ij} + x_{ik} + x_{jk} \leq 2$ defines a facet of $Q(G)$.

A sequence of nodes $v_1, v_2, ..., v_k$ of G is called a *path* if $v_{i-1}v_i \in E$, for $i = 2, ..., k$. Node v_1 is the *origin* and node v_k is the *end* of the path. If $v_1 = v_k$ a path is said to be a *cycle*.

Theorem 2.4. *Let C be a cycle of G with $|V(C)| = p + 1$ then*

$$x(E(C)) \geq 2 \tag{2.5}$$

defines a facet of Q(G).

The inequalities (2.4) and (2.5) are called *clique* and *cycle inequalities*, respectively.

The inequalities (2.4) (and thus the (2.3)) are facet-defining also for the cut polytope (see Barahona and Mahjoub [1]), while the (2.5) are only valid for $Q(G)$ and not for $C(G)$.

The inequalities $x_e \geq 0$ and $x_e \leq 1$ in (2.2) do not define facets of $Q(K_{2p})$, since they can be obtained by nonnegative combinations of the inequalities (2.3). Therefore they could be dropped from the formulation of an LP relaxation of $Q(K_{2p})$. However, we keep them for the technical reason that we want to guarantee that all the linear programs that are solved in the following Procedure 2.1 have a finite optimal solution.

The next lemma shows that the matching inequalities (2.2) which are necessary for description of $PF_p(G)$ are never facet defining for $Q(K_{2p})$. Note that these inequalities are valid for $Q(K_{2p})$ but not for $C(K_{2p})$ and although not facet defining, they are computationally useful and they are used in our algorithm.

Lemma 2.5. *For no subset $W \subseteq V$ and $T \subseteq \delta(W)$, where $p|W| + |T|$ is odd, the inequality*

$$x(E(W)) + x(T) \leq \frac{1}{2}(p|W| + |T| - 1) \tag{2.6}$$

defines a facet of $Q(K_{2p})$.

Proof. By subtracting the linear combination $\frac{1}{2}\sum_{v \in W} x(\delta(v)) = \frac{1}{2}p|W|$ of the equation system of $Q(K_{2p})$ from (2.6), we get the following, wel known, equivalent form for the inequality:

$$x(T) - x(\delta(W) \setminus T) \leq |T| - 1 \tag{2.7}$$

Now it is immediate to see that an equicut $\delta(S)$ satisfies (2.7) at equality if and only if the set $\delta(S) \cap \delta(W)$ is equal either to T minus an edge of T or to T plus an edge of $\delta(W) \setminus T$. Furthermore, for each edge $e \in T$ there exists an equicut $\delta(S)$ such that $\delta(S) \cap \delta(W) = T \setminus \{e\}$. Otherwise $x_e = 1$ for all incidence vectors of equicuts satisfying (2.7) at equality and the face defined by (2.7) is contained in the face defined by $x_e \leq 1$, and so it cannot define a facet. The same argument shows that for each $e \in \delta(W) \setminus T$ there exists an equicut $\delta(S)$ such that $\delta(S) \cap \delta(W) = T \cup \{e\}$.

Assume $T \neq \emptyset$ and let $G' = (V', T)$ be the partial graph of K_{2p}, where V' is the set of endnodes of edges in T.

We first show that G' is connected.

Assume the contrary and let uv and wz be two edges of T, where $\{u, w\} \subseteq W$ and the nodes u and v belong to distinct connected components of G'. Then every equicut that contains $T \cup \{uz\}$ also contains edge wv and therefore cannot satisfy (2.7) at equality. Hence $x_{uz} = 0$ for every equicut satisfying (2.7) at equality, a contradiction.

We now show that G' is either a complete bipartite graph or a complete bipartite graph with onr edge removed.

Assume the contrary. Then either there exist edges uv and wz in T where edges uz and wv are in $\delta(W) \setminus T$ and since G' is connected, the above argument yields a contradiction or there exist edges uv and uw in $\delta(W) \setminus T$, where nodes u, v, w belong to V'. Now since G' is connected, every equicut that contains $T \cup \{uv\}$ also contains uw, a contradiction.

Assume $\delta(W) \setminus T \neq \emptyset$ and let $G'' = (V'', \delta(W) \setminus T)$ be the partial graph of K_{2p} where V'' is the set of endnodes of edges in $\delta(W) \setminus T$.

The same argument as above shows that G'' is a complete bipartite graph or a complete bipartite graph minus one edge.

The two facts imply that either $T = \emptyset$ or T induces a complete bipartite graph, spanning all nodes of W or $V \setminus W$, or $T = \{e\}$ or $T = \delta(W) \setminus \{e\}$. In the first three cases, there exists no equicut satisfying the corresponding inequality at equality. In the last case there is exactly one equicut satisfying the corresponding inequality if $|W| = p$ for $p \geq 2$ and Theorem 2.1 shows that the dimension of $Q(K_{2p})$ is at least 2. ∎

From now on for $Q(K_{2p})$ we use the LP relaxation $\hat{Q}(K_{2p})$ (stronger than $\tilde{Q}(K_{2p})$), described by (2.2), (2.3), (2.4), and (2.5).

2.2. Cutting plane algorithms

Let us denote by $\mathcal{L}_p$ the set of all inequalities describing an LP relaxation $\hat{Q}(K_{2p})$ of $Q(K_{2p})$. To compute a lower bound on the optimal objective function value of (2.1) we have to solve the following linear program

$$\begin{aligned} &\min\ cx \\ &\text{s.t. } A_{2p}x = \mathbf{p} \\ &\quad lx \le l_0 \quad \text{for all } (l, l_0) \in \mathcal{L}_p \\ &\quad \mathbf{0} \le x \le \mathbf{1} \end{aligned} \tag{2.8}$$

where $\mathbf{0}$ and $\mathbf{1}$ are vectors of $\mathbb{R}^E$ with all the components equal to 0 and 1, respectively. Since $|\mathcal{L}_p|$ is finite but generally exponential in n, it is not possible to solve (2.8) by providing an LP optimized with a full description of all inequalities of $\mathcal{L}_p$. Yet (2.8) can be solved by the following general *polyhedral* cutting-plane procedure.

Procedure 2.1. *(Polyhedral Cutting-Plane)*

Input: p, $c \in R^E$, a family of inequalities $\mathcal{L}_p$.

Step 1. Let $\mathcal{L} = \emptyset$.

Step 2. Solve the linear program

$$\begin{aligned} &\min\ cx \\ &\text{s.t. } A_{2p}x = \mathbf{p} \\ &\quad lx \le l_0 \quad \text{for all } (l, l_0) \in \mathcal{L} \\ &\quad \mathbf{0} \le x \le \mathbf{1} \end{aligned} \tag{2.9}$$

and let $\tilde{x}$ be its solution. ($\tilde{x}$ is called the *current LP solution*.)

Step 3. Find one or more inequalities $(l, l_0) \in \mathcal{L}_p$ with $lx > l_0$ (these inequalities are called the *violated inequalities*), or prove that all inequalities in $\mathcal{L}_p$ are satisfied.

Step 4. If no violated inequalities are found, then stop. Otherwise, add the violated inequalities found at Step 3 to $\mathcal{L}$ and go to Step 2.

Procedure 2.1 stops after a finite number of steps because $\mathcal{L}_p$ is finite. Step 3 is called the *separation problem* or the *constraint identification problem*. A procedure that solves Step 3 is said an *exact* separation procedure. We call *heuristic* a procedure that may produce some violated inequalities but that, if does not find any, cannot prove that all members of $\mathcal{L}_p$ are indeed satisfied by the current LP solution. In our algorithm we have exact as well as heuristic separation routines to produce members of $\mathcal{L}_p$ violated by the current LP solution of Step 2. Due to fact that $\hat{Q}(K_{2p})$ is only a relaxation of $Q(K_{2p})$ and due to the heuristic nature of some of the separation procedures, Procedure 2.1 may stop with a solution $\tilde{x}$ which is fractional, and thus cannot be an optimal solution for the problem (2.1). In such a case we proceed by integrating the polyhedral cutting-plane procedure above described with an enumeration procedure. This approach to the problem is the one originally used by Padberg and Rinaldi [12] for the *symmetric traveling salesman problem* and named *branch-and-cut*. We briefly describe the branch-and-cut algorithm using the same notation as in [14] and we refer to the original paper for the details.

For a subset $\mathcal{L}$ of the set $\mathcal{L}_n$ of inequalities that describe a relaxation $\hat{Q}(K_{2p})$ of $Q(K_{2p})$ and for an ordered pair $\langle F_0, F_1 \rangle$ of disjoint edge sets F_0, $F_1 \subseteq E$, by $SP(\mathcal{L}, F_0, F_1)$ we denote the

linear program

$$\begin{aligned} \min\ & cx \\ \text{s.t. } & A_{2p}x = \mathbf{p}, \\ & lx \le l_0 \quad \text{for all } (l, l_0) \in \mathcal{L}, \\ & x_e = 0 \quad \text{for all } e \in F_0, \\ & x_e = 1 \quad \text{for all } e \in F_1, \\ & \mathbf{0} \le x \le \mathbf{1}. \end{aligned}$$

Let $\mathcal{S}$ be a family of ordered pairs of edge sets. Each such pair defines a subproblem produced by the branch-and-cut algorithm. Let x^* be the incidence vector of some equicut of K_{2p} (found, e.g., by a heuristic algorithm).

Algorithm 2.2. *(Branch-and-Cut)*

Input: $\mathcal{L}_p$, c, x^*.

Step 0. *Initialize:* Set $\mathcal{S} = \{\langle \emptyset, \emptyset \rangle\}$, $\mathcal{L} = \emptyset$.

Step 1. *Select the next unsolved subproblem:* If $\mathcal{S} = \emptyset$, then stop. Otherwise pick an ordered pair $\langle F_0, F_1 \rangle$ from $\mathcal{S}$ and replace $\mathcal{S}$ by $\mathcal{S} - \langle F_0, F_1 \rangle$.

Step 2. *Solve LP:* Solve the linear program $SP(\mathcal{L}, F_0, F_1)$.

Step 3. *Fathom a subproblem:* If the LP program is infeasible or if it has an optimal solution $\tilde{x}$ with $c\tilde{x} \ge cx^*$, then go to Step 1.

Step 4. *Generate violated inequalities:* Find one or more inequalities of $\mathcal{L}_p$ that are violated by $\tilde{x}$. If some are found, then add them to $\mathcal{L}$ and go to Step 2.

Step 5. *Better equicut found:* If $\tilde{x}$ is integer, then replace x^* by $\tilde{x}$ and go to Step 1. (Also in this case the subproblem is fathomed.)

Step 6. *Branch:* Pick an edge $e \in E$ such that $0 < \tilde{x}_e < 1$. Replace $\mathcal{S}$ by $\mathcal{S} + \langle F_0 + \{e\}, F_1 \rangle + \langle F_0, F_1 + \{e\} \rangle$ and go to Step 1. (The variable x_e is called the *branching variable*. The two new generated subproblems are called the *children* of the current subproblem $\langle F_0, F_1 \rangle$.)

As for the branch-and-bound, the way Algorithm 2.2 progresses by producing subproblems can be described with a binary tree where each node is associated with a subproblem and every edge describes the relation parent-child between nodes. Such a tree is called the *branch-and-cut tree*. The root of the branch-and-cut tree is the problem solved with Procedure 2.1. Every other node of the tree is associated with a problem like the one solved in Procedure 2.1, but where some variable are *set* to the value 1 and other to the value 0.

The performances of Algorithm 2.1 strongly depend on the total number of generated nodes. The number of nodes is kept low if fathoming at Step 3 applies often. This happens if the value $c\tilde{x}$ (lower bound) is high, i.e., if the relaxation provided by the inequalities in $\mathcal{L}$ is tight, and if the value cx^* of the initial equicut (upper bound) is low, i.e., we provide the algorithm with a 'good' equicut. The first target is reached if the separation problem is solved effectively, the second is attained if an effective heuristic algorithm is executed before Algorithm 2.2.

3. The heuristic algorithm

To find an initial good feasible solution to Problem (2.1) we use a heuristic algorithm. The algorithm is the "exchange" heuristic proposed by Lin and Kernighan [11], which is briefly described here.

Given K_{2p} and $c \in \mathbb{R}^E$, we arbitrarily partition V in two sets A and $B = V \setminus A$ of p nodes and we call (A, B) the *current partition* of V.

Given the current partition (A, B), if we can find two sets $X \subset A$ and $Y \subset B$ with $|X| = |Y| \leq p$, such that $c(\delta(A \setminus X \cup Y)) \leq c(\delta(A))$, then we set

$$\begin{aligned} A' &= A \setminus X \cup Y \\ B' &= V \setminus A' \end{aligned} \tag{3.1}$$

and we declare (A', B') to be the current partition. The operation described in (3.1) is called an *exchange* of the subsets X and Y.

It is possible to decrease the initial $c(\delta(A))$ by a series of exchanges of subsets X and Y; when no further improvement is possible, the current partition is locally minimum with respect to the algorithm. Since there is an exponential number of different ways to choose X and Y, the most critical part of the algorithm is where the sets X and Y are produced.

If two nodes $a \in A$ and $b \in V \setminus A$ are exchanged, they produce a variation $g_1(a, b)$ in the objective function given by $c(\delta(A \cup \{b\} \setminus \{a\})) - c(\delta(A))$. For all pairs in $A \times B$, let (a_1, b_1) be one that minimizes $g(a, b)$. Let $g_1 = g_1(a_1, b_1)$. Then from all $(a, b) \in (A \setminus \{a_1\}) \times (B \setminus \{b_1\})$ we choose the pair (a_2, b_2) that minimizes $g_2(a_2, b_2) = c(\delta(A \cup \{b_1, b_2\} \setminus \{a_1, a_2\})) - c(\delta(A \cup \{b_1\} \setminus \{a_1\}))$ and we set $g_2 = g_2(a_2, b_2)$. We continue until g_p has been computed. If $X = \{a_1, a_2, \ldots, a_k\}$ and $Y = \{b_1, b_2, \ldots, b_k\}$, then the decrease in the objective function, when the sets X and Y are exchanged, is $g_1 + g_2 + \ldots + g_k$. Since $\sum_{i=1}^{p} g_i = 0$, some g_i's are strictly positive and some are strictly negative, unless they are all equal to zero. Then we choose the index k that minimizes the sum $\sum_{i=1}^{k} g_i$. If $\sum_{i=1}^{k} g_i = 0$, the current partition is locally minimum and we stop, if it is negative we exchange the sets X and Y.

4. The constraint generator

The constraint generator is the most important part of our branch-and-cut algorithm. The inequalities produced by the generator fall into one of the following four categories:

(a) matching constraints,
(b) triangle inequalities,
(c) clique inequalities,
(d) cycle inequalities.

The input to the cut generator is the optimal solution $\tilde{x}$ of the current LP relaxation, produced in Step 2 of Procedure 2.1. Depending on the type of inequalities to be generated we represent the current LP solution $\tilde{x}$ in two different ways. One is by its *support graph*, i.e., the weighted graph $(\tilde{G}, \tilde{x}) = (V, \tilde{F}, \tilde{x})$, where $\tilde{F}$ is the subset of E corresponding to all nonzero components of $\tilde{x}$. The weight of each edge $e \in \tilde{F}$ is x_e. The second is by the weighted graph $(G, \tilde{x}) = (V, E, \tilde{x})$, where the weight of each edge $e \in E$ is x_e.

4.1. Separation of matching inequalities

The exact separation of matching inequalities was solved by Padberg and Rao [15] who provide a polynomial time algorithm for the separation problem. This algorithm requires $|\tilde{F}|$ max-flow computations in $G' = (V', F', \tilde{y})$, a graph that is derived from $(\tilde{G}, \tilde{x})$ and has $|V| + |\tilde{F}|$ nodes and $2|\tilde{F}|$ edges.

The Padberg-Rao algorithm is used as an exact separation algorithm of the *2-matching inequalities* of the traveling salesman problem. In this case the support graphs of the LP solutions produced by a polyhedral cutting-plane algorithm are usually very sparse. In addition there exist reduction procedures that can very conveniently applied to these graphs (see, e.g., Padberg and Rinaldi [13]). As a result, the number of max-flow computations necessary to apply the separation algorithm is in general a small fraction of the number of the nodes of the support graph. On the contrary, the support graph in the case of the equicut problem in complete graphs is usually dense and the number of the required max-flow computation is of the order of the square of the number of nodes. Consequently, as the number of node grows, the Padberg-Rao procedure tends to become the bottleneck of the entire constraint generator.

To avoid these difficulties, we use a heuristic procedure proposed by Padberg and Rinaldi (Procedure 4.10 in [13]) that requires only $|V|$ max-flow computations on $(G, \tilde{z})$, where the weight $\tilde{z}$ is derived from $\tilde{x}$. The Padberg-Rinaldi procedure (see [13] for the details) requires that we construct the *cut-equivalent tree* of the Gomory-Hu algorithm associated with $(\tilde{G}, \tilde{z})$ (for a definition of the cut-equivalent tree, see, e.g., [10]). In our implementation such a tree is not produced using the original Gomory-Hu algorithm but its simplified version proposed by Gusfield [9].

Although the Padberg-Rinaldi procedure is not exact, the instances where it fails to find a violated matching constraint seem to be infrequent. To support this feeling, we implemented the exact procedure of Padberg and Rao as well, and we created two versions of our branch-and-cut algorithm that differ only for the procedure that generates violated matching inequalities. We run the two versions on a representative set of 13 instances from our test-bed, with sizes ranging from 20 to 50 nodes. The results of this experiment are reported in Table 1. For the first three labels of the columns of Table 1 and for a description of the instances, see Section 6. For the version of the branch-and-cut code using algorithm a, where a is either Padberg-Rao (`PRAO`) or Padberg-Rinaldi (`PRIN`), `T_`a is total computation time, `C_`a is number of matching inequalities generated, `LP_`a is number of LP solved to reach optimality.

In all tests described in the table the Padberg-Rinaldi procedure runs considerably faster than the exact Padberg-Rao algorithm and never fails to find a violated inequality, since the two algorithms produce the same number of inequalities.

The max-flow algorithm used in both procedures to compute the Gomory-Hu cut-equivalent tree is the one of Goldberg-Tarjan implemented without dynamic tree structure [8].

In our implementation of the constraint generator we add only the first 30 most violated matching inequalities to the $\mathcal{L}$ and we disregard the others.

4.2. Separation of triangle inequalities

The total number of triangle inequalities (2.3) is $4\binom{n}{3}$. Therefore an algorithm that exhaustively produces all of them and checks each of them for violation is exact and runs in $\mathcal{O}(n^3)$ time.

The triangle inequalities (2.3) are of four different types for each triple of distinct nodes of V. For all triples (i, j, ℓ) of nodes we check all the four types of triangle inequalities and then we keep only the most violated of them. Of course, since $0 \leq x_e \leq 1$, for each triple (i, j, ℓ) there

Table 1. *Experiments with different matching separators*

n	NVAR	PERC	T_PRAO	T_PRIN	C_PRAO	C_PRI	LP_PRAO	LP_PRIN
20	190	100	5	2	0	0	6	6
20	190	10	5	3	0	0	6	6
4x5g	190	10	5	3	0	0	5	5
10x2g	190	10	5	3	0	0	7	7
5x4t	190	10	7	4	0	0	7	7
30	435	10	36	20	34	34	10	10
30	435	100	148	76	0	0	19	19
5x6g	435	10	116	67	3	3	20	20
5x6t	435	10	116	65	12	12	18	18
5x8g	780	10	763	559	0	0	30	30
5x8t	780	10	641	365	0	0	31	31
40	780	10	503	187	2	2	31	31
50	1225	10	2084	926	0	0	47	47

is at most one inequality violated. Therefore, once for a given triple we have found a violated inequality of one type we can disregard those of the other types.

In the implementation we add only the first 100 most violated triangle inequalities to $\mathcal{L}$.

4.4. Separation of clique inequalities

Finding a clique inequality violated by $\tilde{x}$ amounts to finding a clique of maximum weight in the graph $(G, \tilde{x})$, which is an $\mathcal{NP}$-hard problem. Thus we devised a heuristic separation procedure for these inequalities.

For each subset $S \subseteq V$, we define an *external cost* $E_a = \sum_{v \in S} \tilde{x}_{av}$ for each node $a \in V \setminus S$ and an *internal cost* $I_b = \sum_{u \in S \setminus b} \tilde{x}_{bu}$ for each node $b \in S$, where $\tilde{x}_{uv}$ is the weight of edge $uv \in E$.

The sets S that we consider are cliques with an odd number of nodes. We start from 3 nodes and go up in size, by adding pair of nodes (the cardinality of the nodes set has to be odd in order to identify a violated constraint of type (2.4)).

Algorithm 4.1. *(Clique Separation)*

Input: $(G, \tilde{x})$, k.

Step 0. Find the triangle $T = (i, j, \ell)$ that maximizes $\tilde{x}(E(T))$; set $q = 3$ and $S = \{i, j, \ell\}$.

Step 1. If $q = 2k + 1$, then stop. Otherwise *Repeat twice:* {add the node $a \in V \setminus S$ with maximum E_a to S}; set $q = q + 2$ and declare all nodes of $V \setminus S$ unflagged and all nodes of S flagged.

Step 2. If $\tilde{x}(E(S)) > \lceil \frac{1}{2}q \rceil \lfloor \frac{1}{2}q \rfloor$, the corresponding constraint is violated, save it.

Step 3. If all nodes are flagged, go to Step 1; otherwise exchange the node b of S with minimum I_b with an unflagged a with maximum E_a; mark a as flagged and go to Step 2.

We have observed that violated inequalities with small value of q are more effective in increasing the objective function value of the current LP. On the other hand, the time consumed by Algorithm 4.1 increases with k. For these reasons, the parameter k is never larger than 5 in our procedure (see Section 4.3).

In the implementation we add only the first 80 most violated clique inequalities to the $\mathcal{L}$ and we disregard the others.

4.4. Separation of cycle inequalities

Finding a cycle inequality violated by $\tilde{x}$ amounts to finding a cycle of prescribed length $(p+1)$ of minimum weight in the graph $(G, \tilde{x})$. This is also a $\mathcal{NP}$-hard problem, since it is as difficult as finding a shortest Hamiltonian cycle in the graph. Therefore, we devised the following heuristic separation procedure for the cycle inequalities.

Algorithm 4.2. *(Cycle Separation)*

Input: $(G, \tilde{x})$, p.

Step 0. Generate a cycle C of $p+1$ nodes using a cheapest insertion algorithm, i.e., start from any node and iteratively add the node connected by the edge of minimum weight to the last added node; mark all nodes in the cycle.

Step 1. Keeping the node set of C unchanged, find a *2-opt* cycle by performing a sequence of *2-exchange* of edges. (A 2-exchange of edges is the operation that starts by removing two non consecutive edges in the cycle, which has weight ℓ, and then adds the two edges that form a new single connected cycle of weight less than ℓ. A 2-opt cycle is one for which no 2-exchange of edges is longer possible.)

Step 2. If $\tilde{x}(E(C)) < 2$, then the corresponding constraint is violated, save it.

Step 3. If all nodes have been marked, then stop; otherwise exchange the node b of C having minimum I_b with the unmarked node a having maximum E_a; mark a and go to Step 1.

In the implementation we add only the first 80 most violated cycle inequalities to the $\mathcal{L}$ and we disregard the others.

4.5. Tailing off

During the process of solving a subproblem $SP(\mathcal{L}, F_0, F_1)$ in Algorithm 2.2, a sequence of monotonically nondecreasing values of objective function values is produced at Step 2. When this sequence is rapidly increasing the algorithm quickly reaches the point when the subproblem can be considered solved either because it is fathomed in Step 3, or because it produces a better integer feasible solution (Step 5), or finally because it generates two new subproblems in Step 6.

It may be sometimes the case that, although some new inequalities, which are violated by the current LP solution $\tilde{x}$, are added to the set $\mathcal{L}$, only a small increase in the objective function value $c\tilde{x}$ is produced. Even worse, sometimes the algorithm may produce a long sequence of iterations of this low quality, before the subproblem can be considered solved because no more inequalities can be generated in Step 4. This behavior of a polyhedral cutting procedure is known as *tailing off* and it may be very inefficient, since each small increase of the objective function is obtained at the non negligible computational costs of generating the inequalities and of solving an LP problem. This behavior is mainly due the heuristic nature of the constraint generator and to the fact that the generator can produce inequalities only from a very restricted number of classes.

To increase the performances of the Algorithm 2.2, when tailing off is detected it is preferable to act in Step 4 as if no inequalities were generated.

Unfortunately, it is not so easy to detect the beginning of the tailing off correctly. Sometimes, after a few steps of small increases of the objective function, the "right" inequalities are generated and the algorithm goes back to a more desirable behavior. Obviously, if we detect the tailing off prematurely, we miss the possibility of increasing the objective function of current subproblem and we unnecessarily increase the number of nodes of the branch-and-cut tree.

To detect tailing off, we introduced a parameter γ that is used as follows: if the increase of $c\tilde{x}$ stays below the parameter γ, then the cut generation is stopped. In the current implementation of the algorithm the default value of γ is $0.0003 \times \pi$, where π is the optimal value of the previous LP relaxation.

4.6. Constraint generation strategy

Although most of the violated inequalities produced by the cut generator are facet-defining for $Q(K_{2p})$, therefore essential to describe it, depending on the objective function c and on the current LP solution $\tilde{x}$, some of them are better than others as they produce a higher increase in the objective function value of the LP current solution. We say that these inequalities are more *effective*. From our computational experiments we found that, on the average, the most effective inequalities are the triangle inequalities, followed by the clique inequalities defined by cliques of small size (5 or 7 nodes). The matching and the cycle inequalities are not effective for unstructured instances for which the components of c are drawn from a uniform random distribution. On the contrary they become effective for structured instance, like those generated on planar or toroidal grids (see Section 6 for their descriptions).

To evaluate the effectiveness of these inequalities, we created two versions of the branch-and-cut algorithm, one with the complete cut generator and the other with the generation of the cycle (or of the matching) inequalities turned off. The first algorithm was up to 5% slower than the second on unstructured instances but up to 30% faster, for example, for instances generated on toroidal grids.

There are two more parameters that we consider to prefer some violated inequality to another. The first is the time taken by the generator and does not need to be commented. The second is the density of the nonzero coefficients of the inequality. In order to solve the LP problems at Step 2 of Algorithm 2.2 fast, we need to keep their constraint matrix as sparse as possible.

To take the different characteristics of the inequalities that we consider into account, we do not execute the separation procedures of the different classes of inequalities all at the same time, but in a hierarchical order. The choice of the hierarchy is motivated primarily by the effectiveness of the inequalities and then by their density and by their generation time. Our separation strategy is summarized by the following procedure.

Procedure 4.3. *(Separation Strategy)*

Input: $\{a_1, a_2, a_3, a_4, a_5\}$, γ (the tailing off parameter), π (the value of the previous LP optimal solution).

Step 1. Solve (2.9) and let $\tilde{x}$ be its solution.

Step 2. If $a_i < c\tilde{x} - \pi < a_{i-2}$ for some $i \in \{3, 5, 7, 9, 11\}$, then execute the triangle separation if $i = 3$ and Algorithm 4.1 with input $k = (i-1)/2$ and $\tilde{x}$, if $i > 3$. Go to Step 4.

Step 3. If $\gamma < c\tilde{x} - \pi < a_{11}$ then execute the generator for the matching inequalities and Algorithm 4.2 (for the cycle inequalities).

Step 4. If any constraints are found, then set $\pi = c\tilde{x}$ and go to Step 1. Otherwise resort to branching (Step 6 of Algorithm 2.2).

In our computational experience the best results were found for $a_3 = 0.009$, $a_i = a_{i-2}/2$ for $i \in \{5, 7, 9, 11\}$ and $a_1 = \infty$.

5. The implementation of branch-and-cut algorithm

Since we are working with complete graphs, the number of variables of the linear programs is $\binom{2p}{2}$ and thus it grows quadratically with the number of nodes. As it seems not feasible to solve instances with more than 100 nodes to optimality with our algorithm, rather than working on a subspace of the variable like it is done in the branch-and-cut algorithm described in Padberg and Rinaldi [14] for the traveling salesman problem, we keep all variables in the current LP formulation.

The first $2p$ rows of the formulation of the current LP are the degree constraints whose density of nonzero entries is $1/p$, since each column has two nonzero entries in the first $2p$ components. Because we solve the LP programs with the bounded Simplex method, the constraints $x \geq \mathbf{0}$ and $x \leq \mathbf{1}$ are not stored explicitly but as lower and upper bounds on the variables. All the other inequalities produced by the cut generator described in Section 4 are added to or removed from the formulation of the current LP with the dynamic mechanism described below.

5.1. The pool

A considerable percentage of the time of our algorithm is consumed by the LP solver (see the tables in Section 6). To reduce this time we try to keep the constraint matrix of each LP program as small as possible. Therefore, after an LP is solved and before adding new violated constraints, we remove all the constraints that are not tight at the current LP solution from the formulation. In our implementation all the inequalities whose current slack value is more than 0.1 are removed. It is then possible that, at some later step, a removed inequality is violated again. However, since the cut generator is not exact, it is also possible that the inequality in not generated a second time. In fact, a heuristic separation procedure may or may not generate an inequality, depending on the LP solution $\tilde{x}$ provided in input. Therefore the method of removing loose constraints provides, in general, shorter LP solution times but weaker final LP relaxation. To avoid the drawback, rather than deleting a loose inequality we store it into a *pool* of loose inequalities. At each iteration of the polyhedral cutting plane procedure and before executing the cut generator, we check if any inequalities of the pool are violated by the current LP solution. If this is the case, these constraints are removed from the pool and added to the set $\mathcal{L}$ of the inequalities of the LP formulation. Some of these inequalities may be produced by the cut generator too, so before adding a newly generated constraint, we check it against duplication. A constraint in the current LP formulation is considered to be tight or loose according to a threshold value of its slack value. If this threshold is too small a constraint may go in and out the pool several times slowing down the convergence of the algorithm. On the other hand, if the threshold is too high the LP basis may grow too much. The value of 0.1 for this threshold has been found as a good compromise after several experiments.

When there is an overflow of the storage area reserved for the pool, the inequalities having large slack value are removed from the pool and forgotten.

When the processing of a subproblem of the branch-and-cut tree is terminated because no more cuts can be generated of because of tailing off, we *mark* all the constraints whose slack variables are out of the current optimal LP basis as undeletable. Marked constraints are never removed from the pool in case of overflow. These constraints are, in general, a subset of the constraints that are tight at the current optimal LP solution and are sufficient to reproduce the best LP solution of the subproblem. Therefore, they should go to the set $\mathcal{L}$ when one of the children of the subproblem is later chosen as the current subproblem. We reconstruct the LP formulation of a parent subproblem before processing its children and the pool permits us to

do the reconstruction efficiently in the following way. Once the variables in F_0 and F_1 are set, the best LP relaxation of the subproblem associated with the pair $\langle F_0, F_1 \rangle$ is reconstructed by running the steps 2, 3, and 4 of Algorithm 2.2, but using only the inequalities of the pool instead of the set $\mathcal{L}_p$.

5.2. Variable fixing and setting

Let $\tilde{x}$ be any basic optimal solution to the linear relaxation of the zero-one program

$$\begin{aligned} &\min\ cx \\ &\text{s.t. } x \in \hat{Q}(K_{2p}) \\ &\quad x \in \mathbb{Z}^E, \end{aligned} \tag{5.1}$$

and let x^* a feasible integer solution of (5.1). As we are solving linear programs with the bounded Simplex method, a component $\tilde{x}_e$ with value 1 can be either a basic variable or a non basic variable at its upper bound. In the latter case its reduced cost r_e is nonpositive. If $r_e < -(cx^* - c\tilde{x})$, then e-th component of every optimal solution to (5.1) has value 1. Analogously, if a component $\tilde{x}_e$ has value 0 and $r_e > cx^* - c\tilde{x}$, then the e-th component of every optimal solution to (5.1) has value 0. In both cases the variable $\tilde{x}_e$ can be permanently *fixed* to its current value.

The inequalities corresponding to the fixing of some variables are not valid for $Q(K_{2p})$ but are satisfied by all the optimal vertices of $Q(K_{2p})$, with respect to the objective function c. Therefore, they can be considered as cutting-planes whose validity depends on the objective function.

Fixing variables reduces the number of variables to be handled and tightens the LP formulation of all the subproblems of the branch-and-cut algorithm.

Let $SP(\mathcal{L}, F_0, F_1)$ any of these subproblems. The variables defined by the two sets F_0 and F_1 are constrained to be equal to 0 and 1, respectively. Since these constraints are only "locally" valid, i.e., are valid only for $SP(\mathcal{L}, F_0, F_1)$ and its "descendants" in the branch-and-cut tree, we say that these variables are *set* to their corresponding values. The reduced costs associated with an optimal basic solution of $SP(\mathcal{L}, F_0, F_1)$ is used to set additional variables in the same way as described before. These settings are valid, of course, only for $SP(\mathcal{L}, F_0, F_1)$ and for its descendants.

Once some variables have been set, either when a subproblem is solved for the first time or when the above reduced cost criterion has been applied, the triangle inequalities (2.3) are used to possibly set more other variables. Let H the subgraph of G induced by the edges corresponding to all the variables that are fixed or set. If two edges are adjacent in H, then the third edge of the triangle that they define can be set using one of the (2.3). If this edge is not in H, then it is added to it. The edge set of H cannot be enlarged any further when all the connected components of H are complete subgraphs.

5.3. The LP solver

For implementing our algorithm we used the routines of the CPLEX CALLABLE LIBRARY, VERSION 2.1 [5]. The CPLEX library offers some features that are helpful in programming a branch-and-cut code, so we decided to make use of them. We give a bound on the objective function to the LP solver: this bound tells it to prematurely stop, when the objective function exceeds the value of the current best equicut. We use the CPLEX AGGREGATOR to use substitution whenever it is possible to reduce the number of rows and we use the default values for all the parameters with which CPLEX can be tuned.

We use both the primal Simplex and the dual Simplex method. Primal Simplex is used to solve the first linear program whose explicit constraints are only the degree equations.

In order to avoid running Phase I of the method we provide the LP solver with a basic primal feasible solution. A basis of the matrix A_{2p} is a well characterized subset of size $2p$ of its columns. The edges corresponding to these columns induce a subgraph of K_{2p}. Each connected component of this subgraph contains exactly one cycle and the length of this cycle is odd. Using this characterization of the basis of A_{2p} and the best partition $(A, V \setminus A)$ found by the heuristic algorithm, we construct a *feasible basis* for the LP solver whose corresponding objective function value is $c(\delta(A))$, i.e., the upper bound found by the heuristic algorithm. To construct such a basis we take the subset of $\delta(A)$ composed by all the edges that connect a given node in A to the nodes of $V \setminus A$ and by all the edges that connect a node in $V \setminus A$ to the nodes of A. Then we add an edge that connects two nodes of A. It is easy to verify that the subgraph of K_{2p} induced by these edges is connected and contains a cycle of length 3. The edges of $\delta(A)$ that are not in the basis are declared to be nonbasic at their upper bound and all the other variables are declared to be nonbasic at their lower bound.

After the first LP program, we always supply the LP solver with the basis of the current LP solution $\tilde{x}$. This basis is always primal infeasible for the next LP program to be solved. In fact, either some constraints violated by $\tilde{x}$ have been added to the LP formulation, or some variables have been set to a value 0 or 1 that does not agree with the value of the corresponding component of $\tilde{x}$. This basis is always dual feasible, though, and so, to avoid executing Phase I of the primal Simplex, we can either solve the LP program with the dual Simplex method or solve the dual of the LP program with the primal Simplex.

Since the efficiency of both the primal and the dual Simplex methods is sensitive to the size of the basis, it is customary to choose the approach that produces a smaller basis (i.e., the first if $|\mathcal{L}| \leq \binom{2p}{2}$, and the second otherwise). In our computational experiments we found, though, that CPLEX is sensitive to the number of variables too. Therefore, to decide which of the two approaches is more convenient to use, we implemented two versions of our branch-and-cut algorithm. Then we run the two versions on the same set of 23 instances taken from our test-bed and having sizes from 20 to 40 nodes. Quite surprisingly, the approach with the dual Simplex is from 2 to 4 times faster than the other. Therefore, we followed this approach in the final implementation of our algorithm.

5.3. Node and variable selection strategies

Two elements that are critical for the efficiency of a branch-and-cut algorithm are the criterion used to select the next unsolved subproblem (Step 1 of Algorithm 2.2) and the criterion used to select the branching variable (Step 6). Due to the good quality of the solutions produced by the heuristic algorithm and to the small sizes of the trees that we generate to solve the instances in our test-bed, we decided to select the simplest among the possible known criteria. Therefore,

we visit the branch-and-cut tree in a "depth first" manner and we pick the variable whose value is the closest to 0.5 as the branching variable.

6. Computational results

The algorithm described in the previous sections was implemented in FORTRAN 77 and tested on randomly generated equicut instances.

Since we did not have instances of the problem coming from real world applications, we created a library of 200 randomly generated instances having sizes from 20 to 70 nodes. These instances fall into four categories.

a) *Pure random instances*: A fixed percentage of the edges (denoted by `PERC` in the following) get weights from 1 to 10 drawn from a uniform distribution. The remaining edges get a 0 weight. Instances with `PERC` less than 100 model the equicut problem on graphs that are not complete. The instances are generated with several values for `PERC`.

b) *Planar grid instances*: To represent instances of equicut on planar grid graphs we assign a weight from 1 to 10, drawn from a uniform distribution, to the edges of a $h \times k$ planar grid, and a 0 weight to the other edges.

c) *Toroidal grid instances*: Same as at point (b) but for toroidal grids, like the one shown in Figure 1.

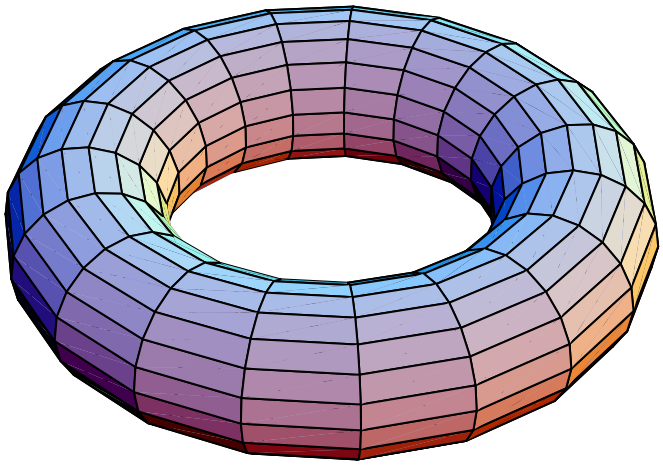

Fig. 1 *A $m \times n$ toroidal grid.*

d) *Mixed grid instances*: These are dense instances with all edges having a nonzero weight. The edges of a planar grid receive weights from 10 to 100 uniformly generated and all the other edges a weight from 1 to 10, also uniformly generated.

Since the objective function for all the instances that we consider is integer, a branch-and-cut subproblem is fathomed (see Step 3 of Algorithm 2.2) when $c\tilde{x} > cx^* - 1$.

For the computational experiments we used several different computers: some DIGITAL machines of the VAX family, a SUN 3/260 workstation, a SUN SPARC 10/41, an IBM RISC 6000 and a NEXT. The computation times that we report are relative to the experiments on the SPARC 10/41 and are expressed in seconds. The functions of the figures 2 and 3 are calculated and plotted with the package MATHEMATICA on a NEXT workstation.

In all tables the size of an instance is given by the number of nodes for the instances of type (a) and by the dimensions of the grid (`h` × `k`) for the instances of the other types. The size of an instance is ended with a '`g`', a '`t`', or a '`m`' for the instances of type (b), (c), or (d), respectively. We also use the following abbreviations:

`n`: number of nodes;
`NVAR`: number of variables;
`PERC`: percentage of nonzero weight edges;
`TLP`: CPU time spent by the LP solver;
`TT`: CPU time consumed by the whole algorithm;
`GAP`: percentage of difference between optimal and heuristic solution;
`CUTR`: total number of triangle inequalities generated;
`CUCL`: total number of clique inequalities generated;
`CUCY`: total number of cycles inequalities generated;
`CUMA`: total number of matching inequalities generated;
`MROW`: maximum number of active inequalities constraints;
`NLP`: total number of LP calls;
`BC`: total number of nodes of the search tree;

In Section 2, we pointed out that triangle and clique inequalities are facet-defining for the max-cut polytope. The triangle inequalities were used in other polyhedral based computational studies for the max-cut problem, such as the ones of Barahona, Grötschel, Jünger and Reinelt [2] or of De Simone and Rinaldi [6], but clique inequalities were not used there. Although De Simone and Rinaldi actually used the hypermetric inequalities, which generalize the clique inequalities, it is possible that a specific routine for the clique inequalities generates inequalities that the more general hypermetric inequalities generator may not identify. Consequently, we decided to use our heuristic procedure for clique inequalities, together with our exact procedure for triangles, also to solve some max-cut instances with the same branch-and-cut algorithm described in the previous sections.

Differently from what is claimed in [2] and [6], triangle inequalities were seldom sufficient to solve our max-cut instances to optimality without resorting to branching. Instead, even with a very simple heuristic procedure for clique inequalities like ours, we were able to solve the same instances at the root node (hence without branching) with a substantially smaller computation time.

The library of test problems is the same used for the equicut problem. By comparing the results of the computations for the max-cut with those for the equicut it seems that, at least for the instances of our library, the equicut is easier to solve than the max-cut problem.

In Table 2 we report the computational results for max-cut using only triangle, or triangle and clique inequalities. Two consecutive rows of the table refer to the same problem instance: the first refers to the algorithm that uses only triangle inequalities and the second refers to the other.

Then we solved all the 200 instances of our library, as equicut problems, to optimality. Rather than reporting the results in a long table of 200 rows, we prefer to list in Table 3 the statistics of a very small representative sample of the instances that we solved.

To give a short report on the total computation time `TT` for all the 200 instances, we computed a least-squares estimate of it, as a function of `n` and of a parameter π. For the instances of type (a), π is the density on nonzero edges `PERC`, for the others, it is the ratio `h/k` of the dimensions of the grids. The least-squares estimate provides the values of the parameters α, β, and γ for

Table 2. *Max-cut instances*

n	NVAR	PERC	TT	TLP	GAP	CUTR	CUCL	MROW	NLP	BC
20	190	90	15	14	0	950		869	37	17
			10	9		925	240	1099	14	1
20	190	30	2	2	0	741		601	8	1
			2	2		741	0	601	8	1
4×5g	190	10	1	0	10	400		400	5	1
			1	0		400	0	400	5	1
4×5t	190	10	2	1	2	721		635	9	1
			2	1		721	0	635	9	1
30	435	100	1774	1768	0	4629		3226	94	35
			313	308	0	2702	2656	4705	75	5
30	435	90	1113	1105	0	3712		2768	97	39
			325	317		2445	3110	5068	45	1
30	435	30	22	20	19	1638		1430	20	3
			7	5		696	320	878	9	3
5×6g	435	10	24	22	19	1638		1430	20	3
			9	5		696	560	878	9	3
5×6t	435	10	64	61	3	1478		1126	17	3
			14	12		1076	1120	1723	12	1
40	780	30	352	342	23	2927		2305	31	3
			150	144		2456	1120	3180	26	1
8×5g	780	10	311	299	27	2327		1970	26	5
			38	34		1566	1360	2295	18	3
8×5t	780	10	616	605	3	2792		1944	30	3
			86	80		1876	1600	2762	20	1

the function that gives the total computation time and that we always assume to have the form

$$\mathtt{TT} = \alpha \mathbf{n}^{\beta} \pi^{\gamma}.$$

The results are reported in Table 4, where, for each type of instances, we also give the number of instances (N) solved to optimality and the value of the error of the estimate (the square root of the sum of the squares of the residuals of all the N instances).

Table 4. *Estimates for the function* TT

Instance type	N	TT	error
(a)	40	$1.0\times10^{-13}\ n^{9.16}\ \pi^{0.83}$	19655.4
(b)	50	$1.2\times10^{-9}\ n^{7.28}\ \pi^{0.04}$	18007.2
(c)	60	$1.5\times10^{-9}\ n^{7.06}\ \pi^{-0.31}$	30048.0
(d)	50	$1.7\times10^{-10}\ n^{7.69}\ \pi^{-0.27}$	10366.2

The functions of Table 4 for the instances of type (a) and (c) are also plotted as a surface in the 3-dimensional space in the figures 2 and 3, respectively.

Table 3. *Relevant equicut instances*

n	NVAR	PERC	TT	TLP	GAP	CUTR	CUCL	CUCY	CUMA	MROW	NLP	BC
20	190	10	3	2	0	591	63	9	0	348	6	1
20	190	20	3	2	0	490	59	9	2	301	5	1
20	190	90	2	1	0	489	0	0	0	334	6	1
20	190	100	2	1	0	496	0	0	0	354	6	1
30	435	10	20	18	0	997	240	42	5	690	10	1
30	435	50	103	98	1	1856	800	0	0	1099	25	1
30	435	90	87	82	0	1749	800	0	0	1239	19	1
30	435	100	73	69	1	1745	640	0	0	1167	19	1
40	780	10	179	170	5	2756	400	57	2	1582	29	1
40	780	20	581	575	1	3823	2000	38	0	1817	48	1
40	780	70	3942	3840	2	4427	4016	0	0	3469	71	5
40	780	80	3303	3278	0	4507	3962	0	0	3310	69	3
40	780	90	3246	3210	0	4613	3848	0	0	3094	70	7
40	780	100	1152	1130	0	3885	2400	0	0	1944	44	1
50	1225	10	958	943	0	5015	480	72	31	2137	51	7
50	1225	20	7225	7218	2	8492	5005	48	10	5050	114	11
50	1225	30	10223	10122	0	8599	5146	0	0	2811	129	1
50	1225	70	20742	20503	0	7552	6460	0	0	5747	119	9
50	1225	90	15602	15204	0	7238	5768	0	0	4854	100	7
50	1225	100	10261	10150	0	7378	5142	0	0	4643	92	5
52	1326	10	2042	2037	0	5146	720	73	3	2317	52	7
54	1431	10	2955	2934	0	7148	720	208	69	2667	73	17
56	1540	10	3143	3125	0	8025	800	216	3	2868	82	17
60	1770	10	10488	10355	5	10871	4160	116	0	5167	113	7
10×2g	190	10	6	5	0	667	204	27	2	409	7	1
5×6g	435	10	61	58	1	1720	320	56	3	1020	19	1
2×16g	496	10	108	103	1	1877	480	89	0	1130	20	3
2×17g	561	10	263	258	0	2158	800	159	8	1491	22	3
18×2g	630	10	262	245	6	2083	720	156	1	1362	22	3
2×19g	703	10	788	755	0	3205	1087	144	10	2271	45	3
5×8g	780	10	851	799	0	2923	230	437	0	1178	30	7
3×14g	861	10	1256	1221	0	3525	1040	258	23	2065	37	5
5×10g	1225	10	2843	2810	0	4376	1360	384	42	2190	44	3
6×10g	1770	10	17994	17978	0	7200	1440	435	0	3043	72	31
4×5t	190	10	3	2	0	397	70	8	0	297	5	1
6×5t	435	10	45	43	4	1371	320	56	0	850	15	1
8×5t	780	10	494	479	0	2947	200	380	7	1218	30	7
20×2t	780	10	681	667	5	2860	800	193	1	1650	30	3
21×2t	861	10	1061	1035	4	3437	960	218	0	1968	36	3
23×2t	1035	10	2788	2755	4	4540	1600	336	0	3317	58	3
4×12t	1128	10	3012	2089	4	4669	1040	293	1	2341	48	5
5×10t	1225	10	2031	2017	0	4300	880	144	0	1821	43	13
10×6t	1770	10	6517	6494	7	8475	2000	440	6	4295	87	3
7×10t	2415	10	28297	28129	0	11387	2080	544	0	4551	114	33
2×10m	190	100	3	2	0	490	141	18	0	329	5	1
6×5m	435	100	28	24	18	1184	160	30	4	691	13	1
3×10m	435	100	30	28	0	991	400	59	5	720	10	1
2×17m	561	100	235	227	0	2031	1280	102	14	1617	21	3
10×4m	780	100	315	307	0	1889	1120	160	0	1406	19	1
5×10m	1225	100	3212	3158	0	3175	2480	399	3	2363	33	5
4×13m	1326	100	5702	5613	0	4255	3040	510	0	3315	43	7
13×4m	1326	100	5105	5043	0	3976	2960	494	0	2812	41	5
10×6m	1770	100	12920	12770	0	4877	3760	746	0	3417	49	9
10×7m	2415	100	127297	126507	0	8663	6240	1454	217	7857	87	13

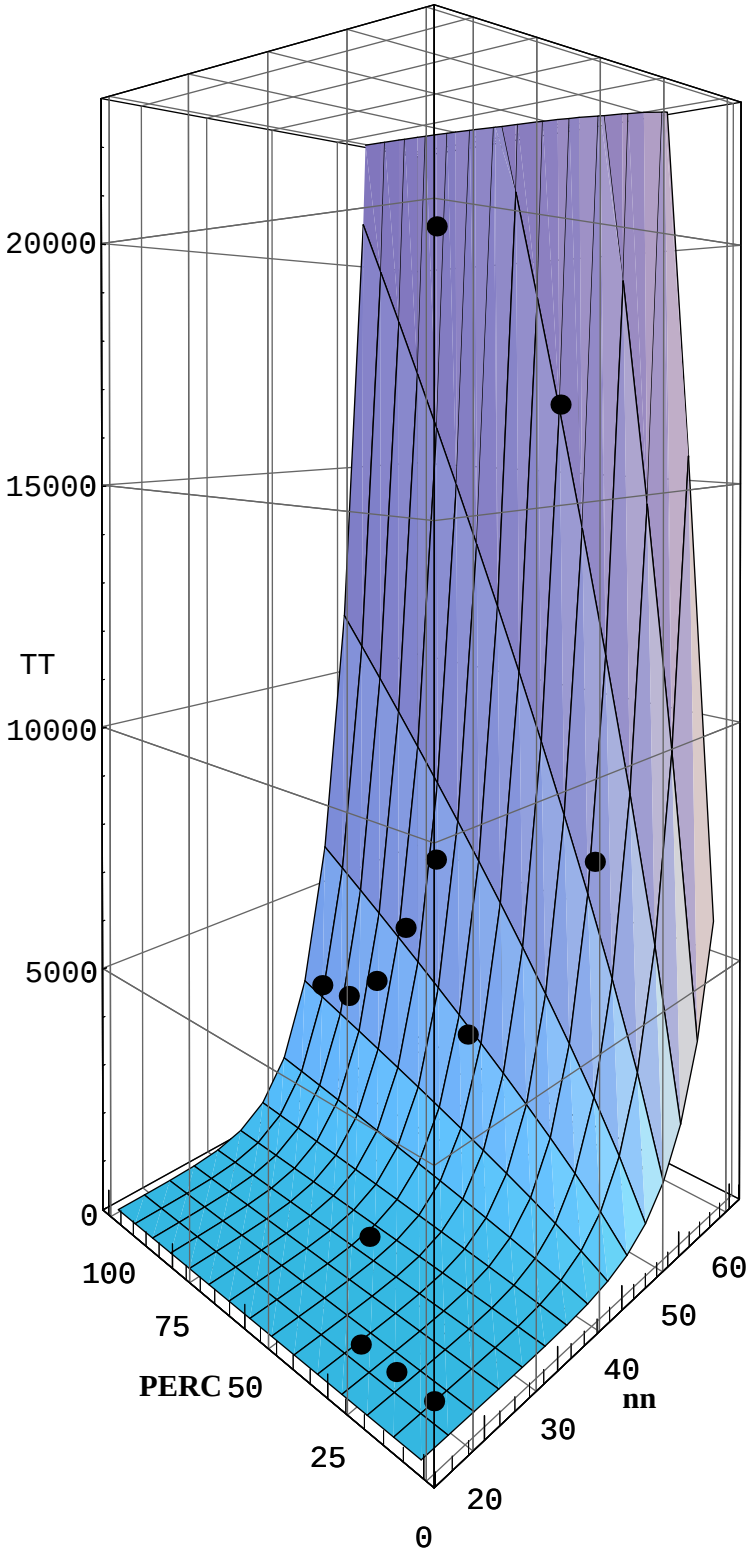

Fig. 2 TT *for pure random instances.*

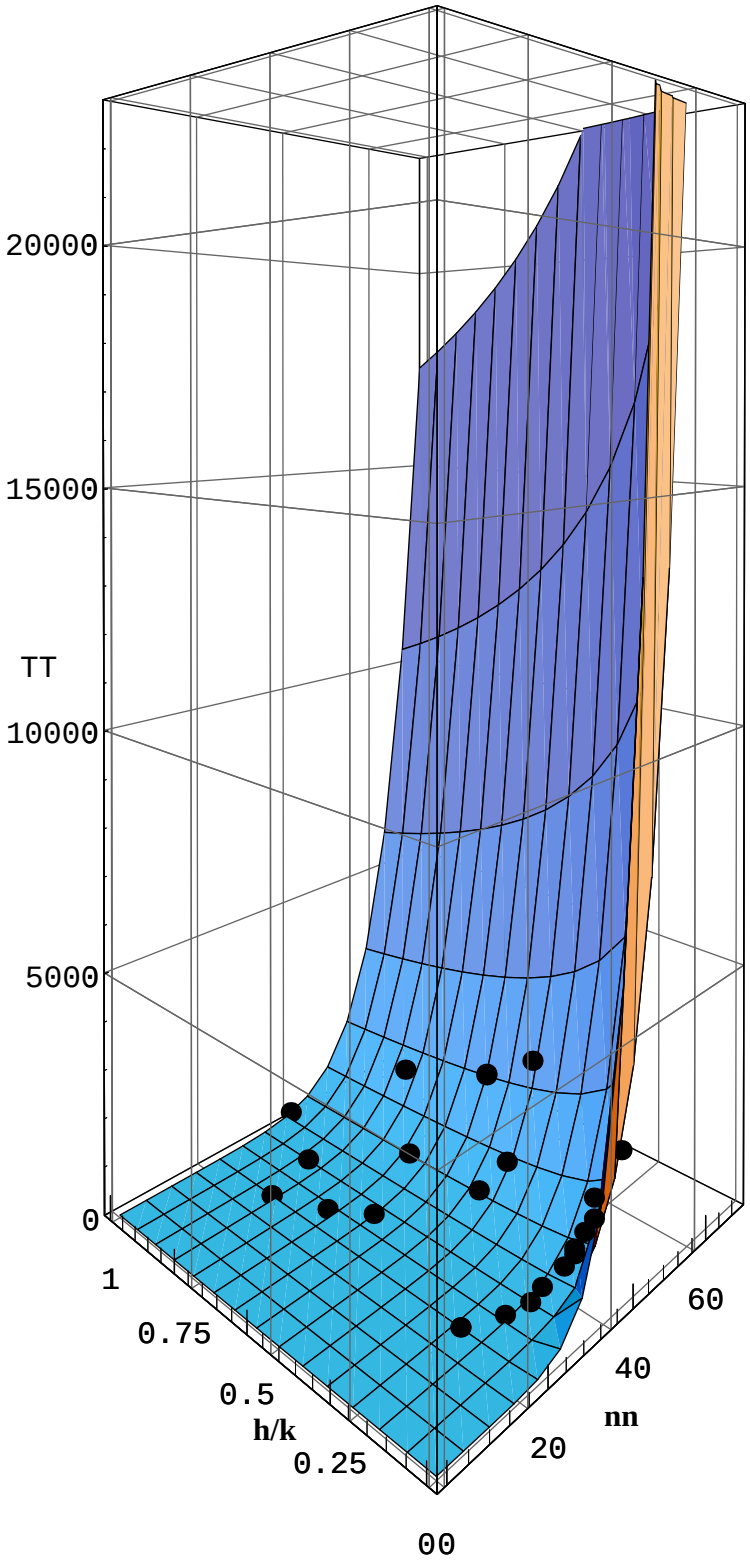

Fig. 3 TT *for toroidal grids.*

Acknowledgments

The research was partially supported by two research grants from *Progetto Finalizzato Trasporti 2* of the Italian National Research Council (contracts CNR/92.018880.PF74 and CNR/93.01812.PF74). Acquisition of CPLEX was made possible by other research grants from *Progetto Finalizzato Trasporti 2* of the Italian National Research Council (contracts CNR/92.01845.PF74 and CNR/93.01778.PF74). We are grateful to Professor Maria Morandi Cecchi for making available to us the workstations IBM RISC 6000, SUN 3/50, and NEXT with their software environments. We would like to thank also Luca Righi for helping us in installing and using some software tools.

References

[1] F. Barahona and Mahjoub, "On the cut polytope," *Mathematical Programming* 36 (1986) 157–173.

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

[3] M. Conforti, M.R. Rao, and A. Sassano, "The equipartition polytope I: Formulations, dimension and basic facets," *Mathematical Programming* 49 (1990) 49–70.

[4] M. Conforti, M.R. Rao, and A. Sassano, "The equipartition polytope II: Valid inequalities and facets," *Mathematical Programming* 49 (1990) 71–90.

[5] CPLEX, "Using the CPLEX Callable Library and CPLEX Mixed Integer Library," CPLEX Optimization Inc. (1992).

[6] C. De Simone and G. Rinaldi, "A cutting-plane algorithm for the max-cut problem," Research Report, IASI CNR (Roma, 1992), to appear in *Optimization Methods and Software*.

[7] J. Edmonds and E.L. Johnson, "Matching: A well solved class of integer linear programs," in R. Guy ed., *Combinatorial Structures and Their Applications* (Gordon and Breach, New York, 1970) pp. 89–92.

[8] A.V. Goldberg, "Efficient graph algorithms for sequential and parallel computers," Working Paper, MIT (Cambridge, Massachusetts, 1992).

[9] D. Gusfield, "Very simple algorithms and programs for all pairs network flow analysis," Research Report, University of California (Davis, California, 1987).

[10] T.C. Hu, "Multiterminal maximal flows," in T.C. Hu, *Integer Programming And Network Flows* (Addison-Wesley, Reading, Massachusetts, 1969) pp. 129–142.

[11] B.W. Kernighan and S. Lin, "An efficient heuristic procedure for partitioning graphs," *Bell Systems Technical Journal* 49 (1970) 291–307.

[12] M. Padberg and G. Rinaldi, "Optimization of a 532-city symmetric traveling salesman problems," *Operations Research Letters* 6 (1987) 1–7.

[13] M. Padberg and G. Rinaldi, "Facet identification for the symmetric traveling salesman polytope," *Mathematical Programming* 47 (1990) 219–258.

[14] M. Padberg and G. Rinaldi, "A branch-and-cut algorithm for the resolution of large-scale symmetric traveling salesman problems," *SIAM Review* 33 (1991) 1–41.

[15] M. Padberg and M.R. Rao, "Odd minimum cut-set and b-matchings," *Mathematics of Operation Research* 7 (1982) 67–80.