Mathematical Programming 78 (1997) 243–263

A branch-and-cut algorithm for the equicut problem

Lorenzo Brunetta[a], Michele Conforti[a,*], Giovanni Rinaldi[b]

[a] *Dipartimento di Matematica Pura e Applicata, Università degli Studi di Padova, via Belzoni 7, 35131 Padova, Italy*

[b] *Istituto di Analisi dei Sistemi ed Informatica del CNR, Roma, Italy*

Received 16 February 1994; revised manuscript received 21 October 1995

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. Finally, we report on computational results on dense instances with sizes up to 100 nodes. © 1997 The Mathematical Programming Society, Inc. Published by Elsevier Science B.V.

Keywords: Equicut; Max-cut; Polyhedral theory; Cutting-plane algorithm; Heuristic algorithm; Branch-and-cut

1. Introduction

We consider an undirected graph $G = (V, E)$, with edge set E, vertex set V, and with edge weights $c_e \in \mathbb{R}$, $e \in E$. By uv we denote 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:

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

The sets S and $V \setminus S$ are called the *shores* of the cut $\delta(S)$. We simply write $\delta(v)$, instead of $\delta(\{v\})$, to denote a cut defined by a single node. A cut is an empty set of

* Corresponding author. Correspondence to: Dipartimento di Matematica Pura e Applicata, Università degli Studi di Padova, via Belzoni 7, 35131 Padova, Italy.

0025-5610/97/$17.00 © 1997 The Mathematical Programming Society, Inc.
Published by Elsevier Science B.V.
PII S0025-5610(97)00017-8

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 $|S| = \lfloor |V|/2 \rfloor$ or $|S| = \lceil |V|/2 \rceil$.

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 a ground state of a *spin glass* having zero magnetization. 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. In numerical analysis it is helpful in finding the $L - U$ factorization of the matrix of a linear system.

The four basic components of our 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 these components in the following sections.

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

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 endpoints 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 a graph G are the convex hulls of the 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:

$$\min \{cx \mid x \in Q(G)\}. \tag{2.1}$$

The optimal solution to (2.1) is the incidence vector of an optimal equicut. Unfortunately, it is unknown 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 $p \geqslant 2$, the dimension of $Q(K_{2p})$ is $\binom{2p}{2} - 2p$ and the affine hull of $Q(K_{2p})$ is $\{x \in \mathbb{R}^E \mid A_{2p}x = \mathbf{p}\}$, where A_{2p} is the $2p \times \binom{2p}{2}$ 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 $p \geqslant 2$.

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 than or equal to p. A p-matching is a *p-factor* if the graph G' is p-regular.

Edmonds and Johnson [8] 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:

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

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]):

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

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 $\max\{cx \mid x\in\tilde{Q}(K_{2p})\}$ 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 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*

$$x(E(K_q))\leqslant\lceil\tfrac{1}{2}q\rceil\lfloor\tfrac{1}{2}q\rfloor \tag{2.4}$$

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

Remark 2.4. From the previous theorem we have, as a corollary, that the triangle inequality $x_{ij}+x_{ik}+x_{jk}\leqslant 2$ defines a facet of $Q(G)$.

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

Theorem 2.5. *Let C be a cycle of G with $|V(C)|=p+1$ then*

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

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 valid for $Q(G)$ but not for $C(G)$.

The inequalities $x_e\geqslant 0$ and $x_e\leqslant 1$ 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 Step 2 of the following Procedure 2.1 have a finite optimal solution.

The next lemma shows that the matching inequalities (2.2), which are necessary for the 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 so they are used in our algorithm. Since $Q(K_4)$ is completely described by the equations and the (2.3), in the lemma we consider only the case $p \geqslant 3$.

Lemma 2.6. *For no $W \subseteq V$ and $T \subseteq \delta(W)$, with $p|W| + |T|$ odd, the inequality*

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

defines a facet of $Q(K_{2p})$, for $p \geqslant 3$.

Proof. Let us assume that (2.6) defines a facet of $Q(K_{2p})$. 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 well known equivalent form for the inequality:

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

An equicut $\delta(S)$ that satisfies (2.7) at equality is called *tight*. It is immediate to see that an equicut is tight 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 a tight equicut such that $\delta(S) \cap \delta(W) = T\setminus\{e\}$. Otherwise, the inequality $x_e = 1$ would be satisfied by the incidence vectors of all tight equicuts and the face defined by (2.7) would be contained in the face defined by $x_e \leqslant 1$, and so it could not define a facet. The same argument, involving the inequality $x_e \geqslant 0$, shows that for each $e \in \delta(W)\setminus T$ there exists a tight equicut such that $\delta(S) \cap \delta(W) = T \cup \{e\}$. The subgraph of K_{2p} induced by the edge set T is the bipartite graph (V_1, V_2, T), where $V_1 = V(T) \cap W$, $V_2 = V(T)\setminus V_1$, and $V(T)$ denotes the set of nodes spanned by T. Without loss of generality we assume $|V_1| \leqslant |V_2|$. Let C be a cycle of (V_1, V_2, T) and e be an edge of C. The cycle C has an even number of edges, and so a tight equicut cannot contain all edges of T but e, because it would intersect C in an odd number of edges. Consequently the graph (V_1, V_2, T) is acyclic.

It is easy to see that $T \neq \emptyset$. Otherwise, a tight equicut would contain only one edge of $\delta(W)$, but this is possible only if $p = 1$.

Suppose now that $T = \delta(W)$. Since (V_1, V_2, T) is acyclic, we have $|W| = 1$ and $|T| = 2p - 1$. For an equicut to be tight, we must have $|\delta(S) \cap \delta(W)| = 2p - 2$ and, since in this case $|\delta(S) \cap \delta(W)| = p$, it follows that $p = 2$. Therefore $T \neq \delta(w)$.

Let e be an edge of $\delta(W)\setminus T$ and $\delta(S)$ be a tight equicut that contains e. Since we must have $T \subseteq \delta(S)$, we can assume, without loss of generality, that $V_1 \subseteq S$ and $V_2 \subseteq (V\setminus S)$. Since e is the only edge of $\delta(W)\setminus T$ that belongs to $\delta(S)$, either $(V_1, V_2, T \cup \{e\})$ or (V_1, V_2, T) is a complete bipartite graph.

In the first case, since (V_1, V_2, T) is acyclic, we must have $|V_1| = |V_2| = 2$. Moreover, $W = V_1$ and $V\setminus W = V_2$, otherwise there would be at least two edges in $\delta(W)\setminus T$ contained in $\delta(S)$. But this implies $p = 2$.

In the second case, since (V_1, V_2, T) is acyclic, $|V_1| = 1$. Let $e = (u, v)$ be the only edge in $\delta(W)\setminus T$ contained in $\delta(S)$ and assume that u is in W. Clearly, $W = V_1 = \{u\}$ and $V\setminus S = V_2 \cup \{v\}$. This implies that $|V_2| = p - 1$. Let $f = (u, w)$ be an edge of T, S be a tight equicut that does not contain f, and assume, again, that u is in W. In this case no edges of $\delta(W)\setminus T$ intersect $\delta(S)$, and so $V\setminus S = V_2\setminus\{w\}$. Hence we must have $|V_2| = p + 1$, a contradiction. □

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).

The following remark about the facial structure of $Q(K_{2p})$ has some computational implications that will be discussed in Section 4.2.

Remark 2.7. Let G be a graph obtained from K_{2p} by removing a nonempty set E' of edges. The incidence vectors of the equicuts of G are obtained from the incidence vectors of the equicuts of K_{2p} by deleting all the entries corresponding to edges in E', that is $Q(G)$ is the *projection* of $Q(K_{2p})$ on the subspace $\{x_e = 0, e \in E'\}$. It is in general difficult to retrieve an algebraic description of the facial structure of a projected polytope $Q(G)$ when such a description is known for $Q(K_{2p})$. So, either one studies the facial structure of $Q(G)$ for *special* classes of graphs (see, e.g., Laurent and De Souza [7] for planar graphs) or, if one wants an algorithm that solves the problem for general graphs and uses valid inequalities for $Q(G)$, then G has to be transformed to a complete graph K'_{2p}, using the construction described in Section 1. We pursue the latter alternative that has the advantage of being more general, but has the disadvantage that problem (2.1) has a big number of variables even if G has only a small number of edges.

2.2. *A cutting plane algorithm*

We describe an algorithm to optimize over an LP relaxation $\hat{Q}(K_{2p})$ of $Q(K_{2p})$. The polytope $\hat{Q}(K_{2p})$ is described by the set $\mathcal{L}_p$ of linear inequalities and by the equations of the affine hull of $Q(K_{2p})$. Since $|\mathcal{L}_p|$ is finite but exponential in p, it is not possible to optimize over $\hat{Q}(K_{2p})$ by solving an LP with a full description of all inequalities of $\mathcal{L}_p$. Yet the optimization can be carried out by the following standard *polyhedral* cutting-plane procedure.

Procedure 2.1. (*Polyhedral cutting-plane*)

Input: p, $c \in \mathbb{R}^E$, a family of inequalities $\mathcal{L}_p$. Step 1. Let $\mathcal{L} = \emptyset$.

Step 2. Solve the linear program

$$\min \{cx \mid A_{2p}x = p,\ lx \leqslant l_0 \text{ for all } (l, l_0) \in \mathcal{L},\ \mathbf{0} \leqslant x \leqslant \mathbf{1}\}, \tag{2.8}$$

where $\mathbf{0}$ and $\mathbf{1}$ are vectors of $\mathbb{R}^E$ with all the components equal to 0 and 1, respectively.

Let $\bar{x}$ be an optimal solution of (2.8) ($\bar{x}$ 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 the 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 algorithms, Procedure 2.1 may stop with a solution $\bar{x}$ 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 described above with an enumeration procedure, i.e., we fix a fractional variable to 0 or 1 and create two new equicut problems that we try to solve with the above described cutting plane approach. This approach to the problem is the one originally used by Padberg and Rinaldi [13] for the *symmetric traveling salesman problem* (TSP) and by them named *branch-and-cut.* We refer to the original paper of Padberg and Rinaldi [15] for a complete description of the method.

In the sections 3 and 4, we describe the details of the separation procedures and of our implementation of a branch-and-cut algorithm for the equicut problem.

A heuristic algorithm is applied at the beginning of the root node of the enumeration tree to find an initial "good" feasible solution to Problem (2.1). The algorithm is the "exchange" heuristic proposed by Lin and Kernighan [12].

3. 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, and (d) cycle inequalities.

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

3.1. Separation of matching inequalities

The exact separation of matching inequalities was solved by Padberg and Rao [16] who provide a polynomial time algorithm for this problem. This algorithm requires $|\tilde{F}|$ max-flow computations in $G' = (V', F', \bar{y})$, a graph that is derived from $(\tilde{G}, \bar{x})$ 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 TSP. 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 be very conveniently applied to these graphs (see, e.g., Padberg and Rinaldi [14]). 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 nodes 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 [14]) that requires only $|V|$ max-flow computations on $(G, \bar{z})$, where the weight $\bar{z}$ is derived from $\bar{x}$. The Padberg–Rinaldi procedure (see [14] for the details) requires that we construct the *cut-equivalent tree* of the Gomory–Hu algorithm associated with $(\tilde{G}, \bar{z})$ (for a definition of the cut-equivalent tree, see, e.g., [11]). In our implementation such a tree is not produced using the original Gomory–Hu algorithm but its simplified version proposed by Gusfield [10].

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 in 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 5. 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's 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 produced 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 [9].

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

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

3.2. 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 $0 \leqslant x_e \leqslant 1$, for each triple (i,j,ℓ) there 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 $\mathcal{L}$.

3.3. Separation of clique inequalities

Finding a clique inequality violated by $\bar{x}$ amounts to finding a clique of maximum weight in the graph $(G,\bar{x})$, 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* $E_a = \sum_{v\in S} \bar{x}_{av}$ for each node $a \in V \setminus S$, and an *internal cost* $I_b = \sum_{u\in S\setminus b} \bar{x}_{bu}$ for each node $b \in S$, where $\bar{x}_{uv}$ is the weight of the 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 node sets has to be odd in order to identify a violated constraint of type (2.4)).

Algorithm 3.1. (*Clique separation*)

```
Input: $(G,\bar{x})$, $k$. Step 0.  Find the triangle $T = (i,j,\ell)$ that maximizes $\bar{x}(E(T))$;
         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 $\bar{x}(E(S)) > \lceil \frac{1}{2}q \rceil \lfloor \frac{1}{2}q \rfloor$, 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 node $a$ 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 the separation algorithm increases with k. For these reasons, the parameter k is never larger than 5 in our procedure (see Section 3.5).

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

3.4. Separation of cycle inequalities

Finding a cycle inequality violated by $\bar{x}$ amounts to finding a cycle of prescribed length $(p+1)$ of minimum weight in the graph $(G,\bar{x})$. 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. In the algorithm we use symbols E_a and I_b with the same meaning as in Section 3.3.

Algorithm 3.2. (*Cycle separation*)

```
Input: $(G,\bar{x})$, $p$. 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 $\ell$, and then adds the two edges that form a new single connected
         cycle of weight less than $\ell$. A 2-opt cycle is one for which 2-exchange of
         edges is no longer possible.)
Step 2.  If $\bar{x}(E(C)) < 2$, 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.

3.5. *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 instances, like those generated on planar or toroidal grids (see Section 5 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% slower, for example, for instances generated on toroidal grids.

There are two more parameters that we consider to prefer some violated inequalities to others. The first is the time taken by the generator and does not require any comment. The second is the density of the nonzero coefficients of the inequality. In order to solve the LP's 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.

In the current implementation of the algorithm the default value of the *tailing off* parameter γ is $0.0003 \times \pi$, where π is the optimal value of the previous LP relaxation (we refer to [15] for a detailed description of tailing off).

Procedure 3.3. (*Separation strategy*)

Input: $\{a_3, a_5, a_7, a_9, a_{11}\}$, γ (the tailing off parameter), π (the value of the previous LP optimal solution). Step 1. Solve (2.8) 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 3.1 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 3.2 (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.

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 $a_1 = \infty$.

4. The implementation of the branch-and-cut algorithm

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 $x \geqslant \mathbf{0}$ and $x \leqslant \mathbf{1}$ 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 3 are added to or removed from the formulation of the current LP with the dynamic mechanism described below.

4.1. The pool

A considerable percentage of the time of our algorithm is consumed by the LP solver (see the tables in Section 5). 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 is not generated a second time. In fact, a heuristic separation procedure may or may not generate an inequality, depending on the LP solution $\bar{x}$ provided in the 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 to 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 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 are set, the best LP relaxation of any subproblem is reconstructed using only the inequalities of the pool instead of the set $\mathcal{L}_p$.

4.2. *Variable fixing and setting*

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

$$\min \{cx \mid x \in \hat{Q}(K_{2p}),\ x \in \mathbb{Z}^E\}, \tag{4.1}$$

and let x^* be a feasible integer solution of (4.1). 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 the e-th component of every optimal solution to (4.1) 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 (4.1) 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.

These fixed variables are constrained to be equal to 0 and 1, respectively. Since these constraints are only "locally" valid, i.e., are valid only for the current subproblem 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 these LP relaxation is used to set additional variables in the same way as described before. These settings are valid, of course, only for the subproblem 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 variables. Let H be 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.

As explained in Remark 2.7 of Section 2.1, in our approach the LP's solved at Step 2 of Procedure 2.1 have $\binom{2p}{2}$ variables. Even though the LP objective function has several zero components (as it is the case when, e.g., the graph of the instance to be solved is sparse) the support graph of any solution of (2.8) is rather dense and only a few edges can be fixed to 0. This is not surprising, because even the support graph of the incidence vector of an equicut is dense, since it has more that half of the edges of the complete graph.

For other combinatorial problems on graphs the situation may be completely different. For example, the support graph of a feasible solution of the TSP has density $\frac{2}{n-1}$. Moreover the density of the support graph of a solution to any of its LP relaxation is not much higher. This nice feature makes it possible to execute a branch-and-cut algorithm for the TSP by solving LP's with only a small subset of the problem variables. All the other variables are implicitly assumed to be at value zero (see, e.g., Padberg and Rinaldi [15]). Such a solution technique is not appropriate for the equicut problem, for which the linear program (2.8) has to be solved explicitly in the space of all the variables. Thus, the LP solver has to face problems that are much larger and harder than those arising in the case of the traveling salesman problem, as it will be clear from the figures of the tables of Section 5.

4.3. 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 $|\mathcal{L}| \leqslant \binom{2p}{2}$, 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.

4.4. 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 and the criterion used to select the branching variable. 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.

5. Computational results

The algorithm described in the previous sections was implemented in FORTRAN 77 and tested on randomly generated as well as on real-world equicut instances. Some real-world instances of equicut problems with sizes from 31 to 100[1] nodes, arising from an application of the finite elements method in fluids (the problem is the $L-U$ factorization of the matrix of the linear system), were kindly provided to us by Cid De Souza. The matrix to be factorized is a band matrix, that is the non-zero elements are located in two bands below and above the main diagonal. These two bands are made up of blocks of non-zero elements that can be moved indifferently below and above the main diagonal. The problem can be modeled as the one of finding a minimum equicut in a planar graph, where the nodes correspond to the blocks and an edge connects two nodes if the corresponding two blocks have non-zero elements in the same row.

We created a library of 250 randomly generated instances having sizes from 20 to 70 nodes[1]. These instances fall into the following five 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 an $h \times k$ planar grid, and a 0 weight to the other edges; our $h \times k$ rectangular grid graphs have hk nodes and $2h - h - k$ edges.

c) *Toroidal grid instances*: Same as at point (b) but for toroidal grids; our $h \times k$ rectangular toroidal graphs have hk nodes and $2hk$ edges.

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.

e) *Instances with negative weights*: Same as point (a), except for the fact that half of the edges get weights from -10 to -1 drawn from a uniform distribution.

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

The computation times that we report are relative to the experiments run on a SPARC 10/41 and are expressed in seconds.

In all tables the size of an instance is given by the number of nodes for the instances of type (a) and (e), and by the dimensions of the grid ($h \times k$) for the instances of the other types. The size of an instance is ended with a 'g', a 't', a 'm', or a 'n' for the instances of type (b), (c), (d), or (e), 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;

[1] accessible via `ftp` anonymous at the ftp site `ftp.math.unipd.it`

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
4x5g	190	10	1	0	10	400		400	5	1
			1	0		400	0	400	5	1
4x5t	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
5x6g	435	10	24	22	19	1638		1430	20	3
			9	5		696	560	878	9	3
5x6t	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
8x5g	780	10	311	299	27	2327		1970	26	5
			38	34		1566	1360	2295	18	3
8x5t	780	10	616	605	3	2792		1944	30	3
			86	80		1876	1600	2762	20	1

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.

Table 3
Real world equicut instances

n	NVAR	PERC	TT	TLP	GAP	CUTR	CUCL	CUCY	CUMA	MROW	NLP	BC
31	496	10	56	53	0	1483	320	0	0	846	15	1
54	1431	5	3218	3186	0	3299	1120	364	0	2030	33	29
60	1770	5	6505	6433	0	5700	1840	523	0	2480	57	37
70	2415	5	5737	5616	0	8500	0	929	515	2973	85	55
74	2701	5	3772	3686	0	3999	1280	569	0	2442	40	33
74	2701	5	27712	27527	0	9498	2240	975	0	4026	95	53
80	3160	4	86140	85815	0	14097	2800	1087	0	4683	141	57
90	4005	3	14659	14443	0	5600	1840	987	0	3697	56	47
100	4950	3	97271	96540	0	11797	4080	2449	0	7184	118	101

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.

In Table 3 we provide the results of our algorithm on eight real world equicut instances. Even if the 100-node instance is one of the largest in size we solved, it is much easier to solve than our random generated ones. It is interesting to note also the large number of cycle and clique inequalities identified by our algorithms for these instances.

Then we solved all the 250 instances of our library, as equicut problems, to optimality. Rather than reporting the results in a long table of 250 rows, we prefer to list in Table 4 the statistics of a very small representative sample of the instances that we solved.

As can be noticed by looking at Table 4, because of the density of the matrix of our formulation and because of the numerical difficulties that CPLEX has to handle, the Simplex algorithm runs very slow while it is solving the LP's, and so the CPU time spent by the LP solver is the bottleneck of the whole algorithm. On the contrary the behavior of the cut generator is quite satisfactory, since it consumes a minute portion of the overall computation time while the number of LP calls and the number of branch-and-cut nodes are small.

To give a short report on the total computation time TT for all the 250 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) and (e), 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 the function that gives the total computation time and that we always assume to have the form

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

The results are reported in Table 5, 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
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	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	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
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
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
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

Table 4—continued

n	NVAR	PERC	TT	TLP	GAP	CUTR	CUCL	CUCY	CUMA	MROW	NLP	BC
30n	435	100	87	84	0	1963	232	0	0	944	34	1
30n	435	90	115	112	0	2146	373	0	0	993	38	1
40n	780	100	3173	3134	0	4418	2687	0	0	1830	152	1
40n	780	60	478	465	12	4729	160	0	0	1419	64	1
40n	780	50	1221	1189	3	6258	1377	0	0	1837	120	1
40n	780	30	688	668	1	5013	625	0	0	1710	88	1
50n	1225	100	96847	96147	0	8659	9008	0	0	4039	749	5
60n	1770	20	4892	4818	0	11948	0	0	0	4012	124	1

Table 5
Estimates for the function TT

Instance type	N	TT	error
(a)	40	$1.0 \times 10^{-13}\, n^{9.16}\, \pi^{0.83}$	19655.4
(b)	50	$1.2 \times 10^{-9}\, n^{7.28}\, \pi^{0.04}$	18007.2
(c)	60	$1.5 \times 10^{-9}\, n^{7.06}\, \pi^{-0.31}$	30048.0
(d)	50	$1.7 \times 10^{-10}\, n^{7.69}\, \pi^{-0.27}$	10366.2
(e)	50	$1.1 \times 10^{-10}\, n^{6.99}\, \pi^{0.4}$	3078.0

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 some workstations available to us. We would like to thank also Luca Righi for helping us in installing and using some software tools.

References

[1] F. Barahona and A.R. 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, *Optimization Methods and Software* 3 (1994) 195-214.

[7] C.C. De Souza and M. Laurent, Some new classes of facets for the equicut polytope, Working Paper, CORE (Louvain-la-Neuve, 1991).

[8] 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) 89-92.
[9] A.V. Goldberg and R.E. Tarjan, A new approach to the maximum flow problem, *Journal of ACM* 35 (1988) 921-940.
[10] D. Gusfield, Very simple algorithms and programs for all pairs network flow analysis, Research Report, University of California (Davis, California, 1987).
[11] T.C. Hu, Multiterminal maximal flows, in: T.C. Hu, *Integer Programming And Network Flows* (Addison-Wesley, Reading, MA, 1969) 129-142.
[12] B.W. Kernighan and S. Lin, An efficient heuristic procedure for partitioning graphs, *Bell Systems Technical Journal* 49 (1970) 291-307.
[13] M. Padberg and G. Rinaldi, Optimization of a 532-city symmetric traveling salesman problems, *Operations Research Letters* 6 (1987) 1-7.
[14] M. Padberg and G. Rinaldi, Facet identification for the symmetric traveling salesman polytope, *Mathematical Programming* 47 (1990) 219-258.
[15] 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.
[16] M. Padberg and M.R. Rao, Odd minimum cut-set and b-matchings, *Mathematics of Operation Research* 7 (1982) 67-80.