The γ-Connected Assignment Problem

Marcus Poggi de Aragão
Departamento de Informática
Pontifícia Universidade Católica do Rio de Janeiro, Brazil
e-mail: poggi@inf.puc-rio.br

Eduardo Uchoa
Instituto de Computação
Universidade Estadual de Campinas, Brazil
e-mail: uchoa@dcc.unicamp.br*

August, 1997
Revised June, 1998

Abstract

Given a graph and costs of assigning to each vertex one of k different colors, we want to find a minimum cost assignment such that no color q induces a subgraph with more than a given number (γ_q) of connected components. This problem arised in the context of contiguity-constrained clustering, but also has a number of other possible applications. We show the problem to be NP-hard. Nevertheless, we derive a dynamic programming algorithm that proves the case where the underlying graph is a tree to be solvable in polynomial time. Next, we propose mixed-integer programming formulations for this problem that lead to branch-and-cut and branch-and-price algorithms. Finally, we introduce a new class of valid inequalities to obtain an enhanced branch-and-cut. Extensive computational experiments are reported.

Keywords:Assignment, Clustering, Cutting, Pricing, Integer Programming

*Current e-mail: uchoa@inf.puc-rio.br

1 Introduction

Suppose we have a graph $G = (V, E)$ with n vertices and m edges, a set $K = \{1, \ldots, k\}$ of colors, a vector $\gamma = (\gamma_1, \ldots, \gamma_k)$ of positive integers, and a $n \times k$ matrix C of rational numbers. Let c_{iq} represent the cost of assigning vertex i with color q. The γ-*CONNECTED ASSIGNMENT PROBLEM* (γ-CAP) is to find a minimum cost assignment of each vertex in V with exactly one color in K, but in a way that no color q induces a subgraph of G with more than γ_q connected components (any such assignment will be called a γ-connected assignment). We describe a real world application as motivation:

Example 1 *A farmer decided to cultivate wheat, corn and soybeans in this season. His large property has already been studied and there is a report that describes small areas of the farm, according to soil composition, topography and humidity. Using this data he can estimate the expected cost (or the expected profit) of growing those cultures in each of the areas. Now he wants an assignment of areas to cultures. The first idea was to plant in each area the culture with minimum cost for that area, which naturally leads to a minimum possible total cost. But in doing so, he realized that the areas planted with the same culture were separated in many small parts within the farm. This is very unconvenient for logistics reasons. For instance, the culture specific machinery would have to be idle while traversing from one part to another. It would be much better, with respect to this point, to assign areas to cultures in such a way that each of the three cultures were united in a single connected part. On the other hand, under such a strong constraint, the total cost can increase too much. So the farmer decided to relax a little this connectivity constraint: he would allow a culture to be separated, but just in a very small number of parts. This relaxation permitted to find a significantly more economic assignment, without complicating too much the logistics.*

The problem in this example can be modeled as a γ-CAP: define G to be a planar graph with a vertex associated to each area of the farm and an edge between each pair of adjacent areas, the colors are the three cultures, c_{iq} is the expected cost of cultivating the culture q in area i and γ_q are the maximum number of connected parts allowed for each culture.

This application of γ-CAP can also be viewed as a variant of the contiguity-constrained clustering problem. Contiguity-constrained clustering requires the entities to be associated to vertices of a graph G, and a partition is valid only if each cluster induces a single (or a small number of) connected component in G. This kind of clustering is often used when the entities are geographical areas and the edges of G usually represent the adjacency between areas. Murtagh [7] presents a survey on considering connectivity when clustering, further analysis can be found in Uchoa [11], where γ-CAP was first formulated. Problems involving partitioning graphs into connected components are most often a hard task to accomplish. An example of such hard problem is presented in Hansen et al. [4].

Another class of γ-CAP applications is when some vertices are already assigned to some color and the problem is to find an assignment of the remaining vertices in order to respect the γ-connectivity.

Example 2 *Suppose the components of a circuit are already placed in a circuit board. The routing problem is to find electrical connections (by copper tracks on the board surface) joining component terminals that have to be linked. Of course, distinct connections must not touch. Moreover, they must be separated by a given minimum distance d to satisfy isolation requirements. Due to manufacturing reasons, the tracks must always run in vertical or horizontal directions. This problem may have no solution. In this case, one may use "jumpers", i.e. additional pieces of wire joining two points in the board. If an unlimited number of jumpers is allowed, the routing problem always have a solution. However, it is not recommended to have too many jumpers, since they increase the noise sensibility and decrease the physical robustness of the circuit. So, one desires to find a solution using a very small number of jumpers.*

This routing problem can also be modeled as a γ-CAP. Superpose to the board a grid where each square cell has a side of length d. Let $G = (V, E)$ be the corresponding grid graph. Assuming the component terminals are always placed over a vertex of V, the sets of terminals that have to be linked can be represented by k disjoint vertex-sets, $T_1, \ldots, T_k$. This kind of discretization of the routing problem is quite traditional (see Lengauer [5]). Now the routing can be solved as a γ-CAP over G, where the colors are the terminal sets and $\gamma_q - 1$ gives the maximum number of jumpers allowed by terminal set. Terminal placement may be represented by setting suitable costs. If i belongs to T_q, $c_{iq} = 0$ and $c_{iq'} = 1$ for each $q' \in K, q' \neq q$. If i does not belong to any terminal set, $c_{iq} = 0$ for each $q \in K$. The solution of this γ-CAP has value 0 iff there is a solution for the routing problem. Given such a solution, the required connections may be obtained by computing a spanning tree for each connected component induced by each color.

This article is organized as follows. Section 2 deals with the theoretical complexity of γ-CAP. A mixed-integer programming formulation is proposed in section 3, which leads to a branch-and-cut algorithm for the problem. Section 4 presents an alternative formulation which leads to a branch-and-price scheme. Section 5 introduces two classes of valid inequalities for γ-CAP, which can significantly improve the algorithms for small values of k. Finally, section 6 presents a set of computational experiences.

2 Problem Complexity

This section studies the complexity of the general γ-CAP. However, actual applications, like the ones described in section 1, should yield more restricted instances. In those cases, G is almost always planar and k is often small. Two important special cases are when G is a grid graph and when G is a tree. Therefore, we also establish the complexity for some of these restricted versions of γ-CAP.

2.1 NP-Complete Cases

We restate γ-CAP as a decision problem. Every possible assignment of the vertices in V to one of the colors, can be interpreted as a partition $P = (P_1, P_2, ..., P_k)$ of the set of vertices. In a feasible partition (corresponding to a γ-connected assignment), the subgraphs induced by each P_q have at most γ_q components. Let $c(P) = \sum_{q\in K}\sum_{i\in P_q} c_{iq}$ be the cost of such a partition.

> **γ-Connected Assignment Problem (γ-CAP)**
> **Instance:** Graph $G = (V, E)$ with n vertices; positive integer k, $n \times k$ matrix C (c_{iq}) of rationals, vector $\gamma = (\gamma_1, \ldots, \gamma_k)$ of positive integers, rational c.
> **Question:** Is there a feasible partition $P = (P_1, P_2, ..., P_k)$ such that $c(P) \leq c$?

Theorem 1 *γ-CAP is NP-complete.*

Proof: Clearly the problem belongs to class NP. In order to prove its NP-completeness, we exhibit a reduction from a vertex weighted version of the Steiner problem on graphs, which is defined as follows.

> **Vertex Weighted Steiner Problem (VWSP)**
> **Instance:** Graph $G' = (V', E')$, subset $S \subseteq V'$, non-negative rationals r_i for every $i \in V' \setminus S$; rational r.
> **Question:** Is there a tree T, which is a subtree of G' that includes all vertices of S, with cost $r(T) = \sum_{i\in T\setminus S} r_i$ at most r ?

The classical Steiner problem on graphs, known to be NP-complete (problem ND12 in Garey and Johnson [2]), has non-negative edge weights. To see that Steiner Problem is still NP-complete even when the non-negative weights are on the vertices (VWSP), consider the following simple reduction: subdivide every edge (u,v) in the graph by introducing a new vertex w with the weight of edge (u,v) and keep the original vertices with weight 0. So, VWSP is NP-complete and we can proceed with our proof.

Set $G = G'$; $k = 2$; $c_{i1} = 0, \forall i \in S$; $c_{i1} = r_i, \forall i \in V' \setminus S$; $c_{i2} = r + 1, \forall i \in S$ and $c_{i2} = 0, \forall i \in V' \setminus S$; $\gamma_1 = 1$, $\gamma_2 = |V'|$; $c = r$. Suppose that VWSP has a "yes" answer. Define partition P as follows. P_1 is the vertex-set of T. P_2 is formed by the remaining vertices. Since P_1 is connected, P is a feasible partition and $c(P) = r(T) \leq r = c$. So γ-CAP also has a "yes" answer. Now suppose that γ-CAP has a "yes" answer. Define T to be any spanning tree of the connected component P_1. Note that since $c(P) \leq c = r < r + 1$, P_1 contains all the vertices of S, and T is a feasible Steiner tree. $r(T) = c(P) \leq c = r$, so VWSP also has a "yes" answer. ∎

This proof shows that γ-CAP is NP-complete even if the costs c_{iq} are all non-negative and even if $k = 2$.

VWSP remains NP-complete even if G' is restricted to be planar, bipartite or a grid graph. This follows from the Steiner Problem with edge costs, which is known to be NP-complete in those particular cases [2]. The reduction is the same one given above for the general case. As the subdivision of all edges in a graph preserves planarity and "bipartiteness", this suffices for these cases. For the grid graph case, adding some extra nodes and edges restores the grid structure. The new nodes receive weights large enough not to appear in any "yes" answer. Therefore, as the proof of theorem 1 preserves the graph, γ-CAP also remains NP-complete for those three cases.

2.2 Polynomial Cases

γ-CAP can be solved in polynomial time by a dynamic programming algorithm when the underlying graph G is a tree and k is fixed. In order to simplify the presentation of the algorithm, we treat a slightly different version of γ-CAP where γ is an integer that stands for a limit on the total number of components, instead of a vector defining the maximum number of components per color. Let us call this version the **Total γ-CAP** problem. Next we show that this algorithm for the Total γ-CAP can be adapted to the original γ-CAP in a straightforward way.

This dynamic programming algorithm works by solving a number of Total γ-CAP subproblems over some subtrees of the original tree G. We begin by choosing any vertex to be the root of G, therefore we give an orientation to G. We can solve the subproblems over the subtrees formed by the leaves of G in a trivial way. Next, subproblems corresponding to subtrees formed by joining a vertex of G and a number of already solved subtrees are solved. These may eventually require the use of another dynamic programming algorithm. The procedure is repeated until the root vertex is reached, when we have a solution to the original Total γ-CAP problem.

ALGORITHM TREE FOR TOTAL γ-CAP

Step 0 Definitions and initialization.

Choose any vertex r of G to be the root of the tree. All the remaining 1-degree vertices are leaves. For each $i \in V, i \neq r$ define $p(i)$ (the predecessor of i) to be the unique vertex adjacent to i that lies in the subpath from r to i. For each $u \in V$, define $S(u)$ (the successors of u) as the set of all vertices i such that $p(i) = u$. Define $T(u)$ (the subtree rooted in vertex u) recursively, as the tree induced by the union of u with the vertices of $T(i)$, for every i in $S(u)$. $T(l) = l$ for each leaf l.

For each $u \in V$, each $q \in K$ and each integer t, $1 \leq t \leq \gamma$, let $H(u,q,t)$ denote the value of the best solution of the problem over $T(u)$, assigning u to color q and using at most t components. Clearly $H(l,q,t) = c_{lq}$ for each leaf l and color q, regardless of the number t.

For each $i \in V, i \neq r$, each $q \in K$ and each integer t, $0 \leq t \leq \gamma - 1$, define $P(i,q,t)$ to be the value of the best solution of the problem over $T(i)$ considering the color of $p(i)$ to be fixed to q and using t more *components in $T(i)$. (Remark that $P(i,q,t)$ is obtained from known values of H and that $P(i,q,0) = H(i,q,1)$ is necessarily the value obtained by assigning all vertices in $T(i)$ to color q).*

Step 1 Iterations.

$P(i,q,t)$ is obtained by the expression:

$$P(i,q,t) = \min[\ \{H(i,q,t+1)\}\ \cup\ \{H(i,q',t) : \forall q' \neq q \in K,\ \ \textit{if } t \geq 1\}\ ]$$

The H solutions of a non-leaf vertex u are calculated by the recursion:

$$H(u,q,t) = c_{uq} + \min_{t_i : \sum_{i \in S(u)} t_i = t-1} \sum_{i \in S(u)} P(i,q,t_i)$$

Step 2 Optimal solution.

The value of the optimal solution is given by $\min_{q \in K} H(r,q,\gamma)$.

This algorithm calculates $O(n.k.\gamma)$ H and P solutions. Computing a single $P(i,q,t)$ solution (supposing all the necessary H solutions are available) takes $O(k)$ time. So, it takes $O(n.k^2.\gamma)$ to calculate all P solutions. Calculating a single $H(u,q,t)$ solution (supposing we already have the necessary P solutions), is trivial when $|S(u)| = 1$. Otherwise we have to solve a classical Resource Allocation Problem, to optimally distribute $t-1$ components among the $|S(u)|$ subtrees, possibly using another dynamic programming algorithm (Bellman and Dreyfus [1]), which takes $O(t^2.|S(u)|)$ (i.e. $O(\gamma^2.n)$). Therefore, calculating all H solutions can be done in $O(n^2.k^2.\gamma^3)$. The overall complexity of this algorithm for the Total γ-CAP on trees is then polynomial over the size of the input, since γ is bounded by n.

To adapt this algorithm to the original γ-CAP problem, we simply have to control and distribute the available number of components per color. This is done by calculating partial solutions H (and P) of the form $H(u, q, t_1, \ldots, t_k)$. Now we have $O(n.k.\prod_{q\in K}\gamma_q)$ such solutions. Computing a single P solution is still done in $O(k)$. But to compute an H solution we may have to solve a Multiresource Allocation Problem by dynamic programming, this costs $O(\prod_{q\in K} {t_q}^2.|S(u)|)$ [1]. Therefore, γ-CAP on trees is polynomial, for a fixed value of k.

A fully detailed description of the Resource and Multiresource Allocation Problems can be found in chapters 1 and 2 of Bellman and Dreyfus [1].

3 A Formulation with an Exponential Number of Constraints

A connected component induced by a color admits a spanning tree, so γ-CAP can be reduced to partition the vertices of the graph into colored forests (all vertices in a forest of color q must be assigned to color q), with at most γ_q trees by each color q forest.

We propose a formulation where those trees are constructed as arborescences (directed trees) on an augmented directed graph defined as follows. Replace every edge $e = (i, j) \in E$ by a pair of opposite arcs, (i, j) and (j, i). Also, add a supervertex named 0, and n new arcs $(0, i)$ from 0 to every $i \in V$. These arcs will be called *root arcs*. Defining A to be the set of all these arcs, we have the augmented graph $G^+ = (V \cup \{0\}, A)$. A connected component induced by a color will now correspond to a colored arborescence on G^+ rooted in vertex 0.

The variables x_{iq}, $i \in V$, $q \in K$ represent whether or not vertex i is assigned to color q. For each arc $a = (i, j) \in A$, and for each color $q \in K$, define additional binary variables y_a^q, where $y_a^q = 1$ iff arc a belongs to an arborescence of color q. The formulation can now be presented:

$$
\text{(F1)}\begin{cases}
\text{Min} & \sum_{i \in V} \sum_{q \in K} c_{iq} x_{iq} & & (1) \\
\text{S.T.} & \sum_{q \in K} x_{iq} = 1 & \forall i \in V & (2) \\
& \sum_{i \in V} y_{0i}^q \leq \gamma_q & \forall q \in K & (3) \\
& \sum_{(j,i) \in \delta^-(i)} y_{ji}^q = x_{iq} & \forall i \in V;\ \forall q \in K & (4) \\
& \sum_{(j,i) \in \delta^-(S)} y_{ji}^q \geq x_{dq} & \forall S \subseteq V,\ 1 < |S| \leq n;\ \forall d \in S;\ \forall q \in K & (5)
\end{cases}
$$

Constraints F1.2 assure x to be an assignment. Constraints F1.3 limit the number of active (with value 1) root arcs by color, therefore controlling the number of components per color. Constraints F1.4 imply that if a vertex i is assigned to color q, there must be exactly one active arc entering i. Constraints F1.5 say that if a vertex i is assigned to color q, there must be a path of active arcs of color q from 0 to i. Moreover, F1.4 and F1.5 also restrict an arc of color q to be active only when both of its endpoints are assigned to color q. To see this, let $e = (i, j)$ be an edge of E, q be a color and define $S = \{i, j\}$. Adding the two equalities of F1.4 corresponding to vertices i,j and color q we get $y_{ij}^q + y_{ji}^q + \sum_{(j,i) \in \delta^-(S)} y_{ji}^q = x_{iq} + x_{jq}$. Next, subtract the inequality of F1.5 corresponding to S, $d = j$ and q to get $y_{ij}^q + y_{ji}^q \leq x_{iq}$. Therefore, a binary solution (x, y) respecting F1 implies x to be a γ-connected assignment. Remark that only x really needs to be binary.

We have an exponential number of constraints F1.5. Despite of that, they can be separated in a rather straightforward way. We apply a min-cut algorithm on the augmented graph, from 0 to each vertex $d \in V$ and for each color $q \in K$, using the current values of the arc variables of color q as the arc capacities. If the value of the min-cut is less than the current value of x_{dq}, we have found a violated constraint of F1.5. Therefore, at most $k.n$ such min-cut problems need to be solved to verify all F1.5 constraints to be satisfied, i.e. the separation problem has polynomial complexity.

Magnanti and Wolsey [6] (chapter 6) describe several alternative formulations for the vertex weighted Steiner problem. Many of those suggest other possible ways to enforce the γ-connectivity. This reference contains a wealth of informations on formulations of problems that explicitly or implicitly involve trees.

The above formulation leads to a branch-and-cut algorithm for γ-CAP. Figure 1 illustrates a fractional solution of the linear relaxation of F1. We use the following conventions to present fractional solutions of some instances with $k = 2$. A vertex i colored black means $x_{i1} = 1$ (q=1 stands for black). A white vertex i means

$x_{i2} = 1$ (q=2 stands for white). A grey vertex i means $x_{i1} = 0.5$ and $x_{i2} = 0.5$. If a value near an arc a is a fraction (1/2, for instance), this is the value of y_a^1. Else, it is in decimal format and stands for the value of y_a^2. The root arcs are represented with loose tail and the arcs with 0 value are not shown.

4 A Formulation with an Exponential Number of Columns

The lower bound given by formulation F1 is often tight and it allows the branch-and-cut to solve many instances of reasonable size. However, for harder instances the bound is not so tight, and the algorithm may take an excessive amount of time (section 6). We describe an alternative formulation that may give tighter bounds.

Let L be the set of the incidence vectors of all possible connected components of G along with a tag identifying the color of the component. This means that for each $l \in L$, $l_i = 1$ iff vertex i is in component l, and $l_{n+q} = 1$ iff l is a color q component, so l is a $n + k$ vector. Let $c_l = \sum_{i:l_i=1,q:l_{n+q}=1} c_{iq}$ be the cost of this component. Let λ_l be a binary variable associated with component l. We can formulate γ-CAP as the following cardinality constrained set-partitioning problem.

$$(F2)\left\{\begin{array}{llll} \text{Min} & \displaystyle\sum_{l\in L} c_l\lambda_l & & (1) \\ \text{Subject to} & \displaystyle\sum_{l\in L} l_i\lambda_l = 1 & \forall i \in V & (2) \\ & \displaystyle\sum_{l\in L} l_q\lambda_l \leq \gamma_q & \forall q \in K & (3) \end{array}\right.$$

The exponential number of columns in F2 demands a column generation approach. Let π_i and μ_q be the current values of the dual variables associated with constraints F2.2 and F2.3 respectively. We have k pricing problems, one for each color. Problem q can be stated as follows:

$$\begin{array}{ll} \text{Min} & \displaystyle\sum_{i\in V}(c_{iq} - \pi_i)w_i - \mu_q \\ \text{Subject to} & w \text{ is connected} \end{array}$$

where w_i is a binary variable deciding whether vertex i belongs to a color q component. These pricing problems can also be stated as follows:

Minimum Weight Connected Subset (MWCS): *Given a graph $G = (V, E)$ and weights p_i for each vertex $i \in V$, find a subset S of connected vertices minimizing $\sum_{i\in S} p_i$.*

The NP-hardness of MWCS can be proven by the following reduction from the VWSP, described in section 2. Set the weight of the vertices in R to a negative number large enough to assure that this vertex will be in any optimal solution. The vertices in $V \setminus R$ keep their original non-negatives weights. This result does not mean that column generation can not be carried out. Most of the time the pricing can be performed by fast heuristics. Only when the heuristic does not find a negative reduced cost column, we have to use an exact pricing. Moreover, this exact pricing is relatively easy when compared to the original γ-CAP problem.

Formulation F2 can be viewed as a Dantzig-Wolfe decomposition of F1. Take F1.2 as the coupling constraints and note that F1.3-5 define k independent subproblems. Further divide each such subproblem into γ_q independent, but equal ones. Since we only admit integral solutions to these subproblems, the resulting formulation is necessarily stronger (Geoffrion [3]).

Figure 2 presents a fractional solution of the linear relaxation of F2. The vertices enclosed inside dotted lines define two black components, each of them with value 0.5. Two white components are inside the full lines, also with value 0.5. This fractional solution is exactly equivalent to the fractional solution of F1 presented in figure 1. In other words, for this instance the lower bounds given by F1 and F2 coincide. Computational experiments have shown this to arrive in almost all tested instances. Figure 3 presents a example of a fractional solution to F1 that cannot be represented in F2, i.e. in this case F2 may give a tighter lower bound.

To find optimal integer solutions, we embed F2 in a branch-and-price scheme. Observe the branching can be done on the original x variables. Setting x_{iq} to 0 amounts to remove vertex i from subproblem q. On the other hand, when x_{iq} is set to 1, we remove vertex i from all subproblems with color different from q. Therefore the pricing problems structure remain unchanged after every branching.

5 A Class of Valid Inequalities

Formulation F2 did not yield, in practice, the expected improvement on the lower bounds. Another effort in this direction follows. Consider the fractional solution of F1 presented in figure 1. Observe that any path from vertex a to b in the underlying graph G separates c from d (and vice-versa). As $\gamma_1 = \gamma_2 = 1$, the following inequality is valid for this instance:

$$x_{a1} + x_{b1} + x_{c2} + x_{d2} \leq 3$$

Definition 1 *Let (s_1, t_1) and (s_2, t_2) be two pairs of distinct vertices in V with the following property: there are no two vertex-disjoint paths joining s_1 to t_1 and s_2*

to t_2. Let q_1 and q_2 be two distinct colors in K, with $\gamma_{q_1} = \gamma_{q_2} = 1$. The following valid inequality will be called a **cross inequality***:*

$$x_{s_1\,q_1} + x_{t_1\,q_1} + x_{s_2\,q_2} + x_{t_2\,q_2} \leq 3$$

The cross inequalities can be separated in polynomial time, since deciding whether two vertex disjoint paths between two pairs of vertices exist can be done in $O(n.m)$ time (Shiloach [9]). Moreover, when the underlying graph is planar this task can be accomplished in $O(n+m)$ time, most of the time it amounts to just check if there exists a face of the graph where these pairs alternate in the cyclic order (in figure 1 this is the outer face). As the number of possible two pairs is clearly polynomial, the separation polynomiality follows. An enhanced version of the branch-and-cut algorithm over F1 were obtained by attaching an efficient exact separation procedure for the cross inequality.

This class of inequalities can be generalized to be valid for any γ-CAP instance, with no restrictions on the γ_q values.

Definition 2 *Let S be any subset of V. Let (s_1, t_1) and (s_2, t_2) be two pairs of distinct vertices in S with the following property: there are no two vertex-disjoint paths in S joining s_1 to t_1 and s_2 to t_2. Let q_1 and q_2 be two distinct colors in K. The following valid inequality will be called a* **generalized cross inequality***:*

$$x_{s_1\,q_1} + x_{t_1\,q_1} + x_{s_2\,q_2} + x_{t_2\,q_2} \leq 1 + \sum_{(j,i)\in\delta^-(S)} y_{ji}^{q_1} + \sum_{(j,i)\in\delta^-(S)} y_{ji}^{q_2}$$

First remark that when $S = V$ and $\gamma_{q_1} = \gamma_{q_2} = 1$ this inequality reduces to the simple cross inequality, as the first and the second summations on the right-hand-side are bounded by 1 (by F1.3). Generally these inequalities can be interpreted as follows: if there are no disjoint paths in S linking the two pairs, one pair can possibly be connected through S, but then the other pair can only be connected by a path using vertices of V outside S, or yet, by using an additional root arc. In either case, if the left-hand-side is 4, at least 3 active arcs of classes q_1 or q_2 must enter S.

Even if $\gamma_q = 1$ for each color q, the generalized cross inequalities can be useful, i.e. they may be violated when the simple ones are not. Consider the fractional solution of F1 presented in figure 4. This instance has $k = 3$ and vertex e is assigned to a third color. Taking S as the vertices inside the circle, there are no two disjoint paths in S joining a to b and c to d (although they do exist in V). Since there are just 2 active arcs entering S, we have a violated generalized cross inequality. In this particular case, this inequality captures the fact that if a vertex is to be assigned to a color other than q_1 and q_2, no joining path can use it.

Even if $\gamma_q = 1$ for each color q and $k = 2$ the generalized cross inequalities can still be useful. Consider the fractional solution of F1 presented in figure 5 and let S be the encircled vertices. As there are no two disjoint paths in S joining a to b and c to d, the left-hand-side is 3.5 and the sum of the values of arcs entering S is just 2, we have a violated inequality.

Since no separation procedure were implemented for the generalized cross inequality, its practical strength is still to be measured.

6 Computational Experience

We generate test instances as follows. Costs c_{iq} were picked from a random uniform distribution of integers in the range 0 to 99. Four types of graph G were considered:

1. **Square grid graphs** of size $S \times S = n$. These graphs have $m = 2(n - S)$ edges.
2. **Random planar graphs** generated by the function *random_planar_graph* provided by LEDA library [8]. These graphs are constructed to have about the same number of edges of a square grid graph with n vertices.
3. **Random connected "sparse" graphs** (non-planar) generated in the following way. First, we build a complete graph on n vertices, attribute a random weight to each of its edges and compute a minimum weight spanning tree, therefore generating a random tree. Then we add random edges to get the same number of edges of a square grid graph with n vertices.
4. **Random connected "dense" graphs** (non-planar) generated by computing a random tree and then adding random edges until we have $5.n$ edges.

The branch-and-cut and branch-and-price algorithms were implemented in C++, with the help of the ABACUS framework [10] . The LPs were solved by CPLEX 3.0. The times were obtained on a Sun ULTRA workstation.

Each entry in the tables corresponds to running a set of 5 instances with the specified parameters. **Tm** is the average time for this set of instances in seconds of cpu. As we set a limit of 900 seconds for each run, we can not calculate this average if some of this instances exceeds this time limit. **T** is the median time for the set of instances. Even when one or two instances exceed the limit, we still can calculate the median. **N** is the number of nodes opened in the branch-and-bound tree for this median instance. **LP** is the number of solved LPs for the median instance. As the performance of a typical branch-and-bound can variate wildly, even with similar instances, we believe that median information is more representative of the expected behavior of the algorithm.

The first three groups of experiences (tables 1-3) are aimed at estimating how the problem difficulty changes along with graph type and with its parameters n, k, γ_q (which is the same for every color in all experiments). This is done by reporting the results obtained with our branch-and-cut implementation over F1 (without adding cross inequalities).

Table 1 shows clearly that γ-CAP becames much easier as γ_q grows. This behavior was quite expected, since raising the γ_q means that the optimal solution can be closer to the unconstrained solution given by assigning each vertex i to the color of minimum c_{iq}. So, less constraints of (F1.5) need to be separated, and the lower bound given by F1 is usually much tighter. When $\gamma_q = 1$, solving γ-CAP can become quite hard, even for moderate sizes of n.

Table 2 shows how γ-CAP changes with the number of colors. As we increase k, formulation F1 gets bigger and it takes more time to solve an LP. On the other hand, the lower bound given by F1 is usually tighter. When the size of n is small, it is not clear which are the harder cases. But as n grows, it becomes more important to have a tighter formulation and instances with small values of k seem harder.

Table 3 presents experiments with different types of graph. It suggests the difficulty to grow in the order: grid, random planar, random sparse, and random dense. The last two being clearly much easier than the former ones, which have nearly the same number of edges of the random sparse graph. This seems to reveal that γ-CAP is harder when the underlying graph G is planar (or near-planar), not too sparse and not dense.

The following group of experiments (tables 4-6) compares the branch-and-cut over F1, used in the first 3 tables, with the branch-and-price over F2 and the enhanced (with the cross inequalities) branch-and-cut over F1.

Table 4 depicts the difficulties our branch-and-price algorithms had in solving most of the instances we tried it on (compare with table 1). The point is that solving the subproblem corresponding to a branch-and-bound node is harder for F2 than for F1. The bottleneck of our column generation scheme is the exact solution of the pricing, at least k (usually more) such exact pricings have to be done per node. As the lower bounds given by F2, over all the test instances, were practically the same ones given by F1, the branch-and-cut and branch-and-price trees have about the same size. This explains the poorer performance of the branch-and-price over F2.

Tables 5 and 6 show a remarkable improvement on solving γ-CAP when $k = 2$ due to the strength of the cross inequalities. When $k = 3$ or $k = 5$, a less impressing improvement could be observed, although it was a significant one. For larger values of k only slight improvements were observed. A reason for this is the fact that the simple cross inequalities are seldom violated in these cases. We expect the generalized cross inequalities cover this gap. Table 5 compares with table 2 and second column of table 6 compares with second column of table 3.

Finally, it is interesting to note that, for this algorithm and $k = 2$, the instances on random planar graphs became harder than the ones on grid graphs.

Acknowledgments: This research has been partially supported by CNPq (Project ProComb of ProTeM-II-C – Processo 68006594/94-6) and by Fundação de Amparo à Pesquisa do Estado de São Paulo (FAPESP) (Projeto Temático: Laboratório de Algoritmos Combinatoriais). The first author was also supported by CNPq Bolsa de Produtividade em Pesquisa – Processo 350265/94-1 and by Fundação de Amparo à Pesquisa do Estado do Rio de Janeiro (FAPERJ) – Processo E26/1700667/97. The second author was also supported by CNPq - Bolsa de Mestrado - Processo 830.231/95-1 and by FAPESP - Bolsa de Mestrado - Processo 96/0883-2.

References

[1] Bellman, R., and Dreyfus, S., *Applied Dynamic Programming*, Princeton University Press, New Jersey, 1962.

[2] Garey, M., and Johnson, D., *Computers and Intractability: A guide to the theory of NP-completeness*, Freeman, San Francisco, 1979.

[3] Geoffrion, A., "Lagrangian Relaxation for Integer Programming", *Mathematical Programming Study 2*, 1974, 82-114.

[4] Hansen, P., Jaumard, B., Simeone, B., and Doring, V., "Maximum Split Clustering Under Connectivity Constraints", Les Cahiers du GERAD, G-93-06, Montréal, 1993.

[5] Lengauer, T., *Combinatorial Algorithms for Integrated Circuit Layout*, John Wiley & Sons, Chichester, 1990.

[6] Magnanti, T., and Wolsey, L., "Optimal Trees", CORE Discussion Paper 9426, Université Catholique de Louvain, Louvain-la-Neuve, 1994.

[7] Murtagh, F., "A Survey of Algorithms for Contiguity-Constrained Clustering and Related Problems", *The Computer Journal*, 28-1, 1985.

[8] Näher, S., and Uhrig, C., *The LEDA User Manual Version R 3.3.c*, Max-Planck-Institut für Informatik, Saarbrücken, 1994.

[9] Shiloach, Y., "A Polynomial Solution to The Undirected Two Paths Problem", *Journal of the ACM*, Vol. 27, No.3, Julho 1980.

[10] Thienel, S., "ABACUS - A Branch-And-Cut System", PhD thesis, Universität zu Köln, 1995.

[11] Uchoa, E., "Problemas de Classificação com Restrições de Conexidade Flexibilizadas", Master Dissertation, Instituto de Computação - Universidade Estadual de Campinas, Campinas, 1997.

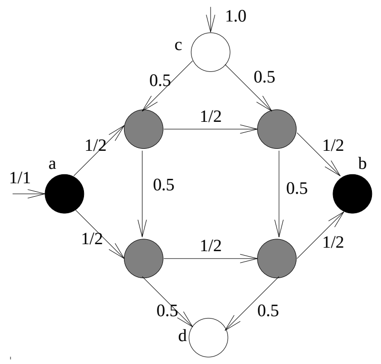

Figure 1: A Fractional Solution of F1 ($k = 2$, $\gamma_1 = 1$, $\gamma_2 = 1$).

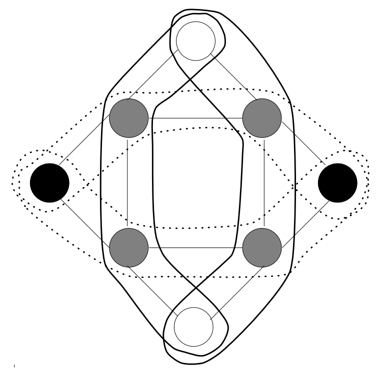

Figure 2: A Fractional Solution of F2 ($k = 2$, $\gamma_1 = 1$, $\gamma_2 = 1$).

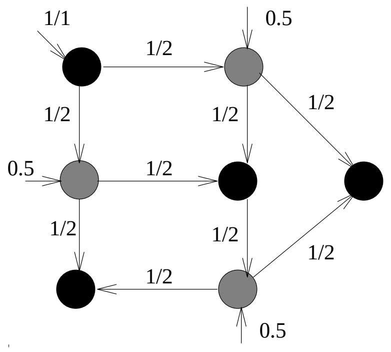

Figure 3: A Fractional Solution of F1 ($k = 2$, $\gamma_1 = 1$, $\gamma_2 = 2$).

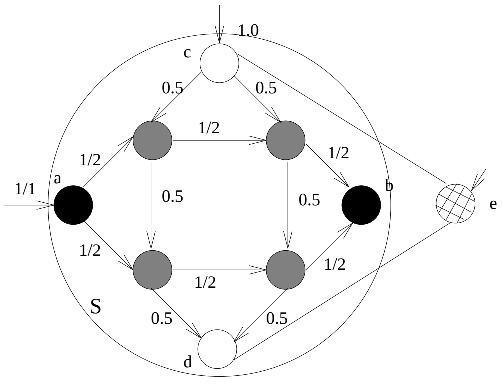

Figure 4: A Fractional Solution of F1 ($k = 3$, $\gamma_1 = 1$, $\gamma_2 = 1$, $\gamma_3 = 1$).

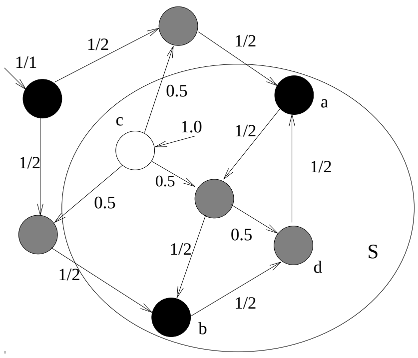

Figure 5: A Fractional Solution of F1 ($k = 2$, $\gamma_1 = 1$, $\gamma_2 = 1$).

n	$\gamma_q = 1$				$\gamma_q = 2$				$\gamma_q = 3$			
	Tm	T	N	LP	Tm	T	N	LP	Tm	T	N	LP
25	3.6	2.7	5	22	1.1	0.3	1	2	0.2	0.2	1	2
36	15	13	23	59	3.0	2.6	5	12	1.8	0.5	1	2
49	-	186	111	300	25	14	11	36	5	1.5	1	6
64	-	-	-	-	98	43	11	46	15	16	9	28
81	-	-	-	-	362	327	47	233	37	36	11	27

Table 1: $k = 3$, grid graph, branch-and-cut over F1.

n	$k = 2$				$k = 3$				$k = 5$				$k = 10$			
	Tm	T	N	LP	Tm	T	N	LP	Tm	T	N	LP	Tm	T	N	LP
25	2.2	1.4	11	15	3.6	2.7	5	22	2.5	1.0	1	8	4.5	2.1	1	3
36	41	18	37	131	15	13	23	59	51	35	31	81	24	15	5	13
49	187	133	111	531	-	186	111	300	170	183	63	175	70	62	7	21
64	-	-	-	-	-	-	-	-	-	-	-	-	-	873	51	160

Table 2: $\gamma_q = 1$, grid graph, branch-and-cut over F1.

n	Grid				Random Planar				Random Esparse				Random Dense			
	Tm	T	N	LP	Tm	T	N	LP	Tm	T	N	LP	Tm	T	N	LP
25	3.6	2.7	5	22	1.0	0.9	1	11	0.5	0.4	1	5	1.5	1.0	1	2
36	15	13	23	59	16	13	23	53	2.3	1.9	3	8	2.1	1.7	1	2
49	-	186	111	300	75	64	33	162	7.3	4.8	1	10	4.0	4.0	1	2
64	-	-	-	-	-	761	179	1077	17	5.6	1	21	39	19	1	24
81	-	-	-	-	-	-	-	-	52	38	1	32	48	10	1	2

Table 3: $k = 3$, $\gamma_q = 1$, branch-and-cut over F1.

n	$\gamma_q = 1$				$\gamma_q = 2$				$\gamma_q = 3$			
	Tm	T	N	LP	Tm	T	N	LP	Tm	T	N	LP
25	15	10	3	69	3.8	2.4	1	25	1.9	1.1	1	20
36	292	118	19	205	23	6	3	46	7.3	3.9	1	23
49	-	-	-	-	175	96	11	109	43	13	1	74
64	-	-	-	-	-	-	-	-	-	196	11	162

Table 4: $k = 3$, grid graph, branch-and-price over F2.

n	$k=2$				$k=3$				$k=5$			
	Tm	T	N	LP	Tm	T	N	LP	Tm	T	N	LP
25	0.6	0.4	1	6	2.9	2.7	3	27	2.1	0.8	1	3
36	2.5	1.0	1	9	9.1	8.5	3	43	51	39	27	80
49	6.0	3.7	1	23	160	164	45	437	180	148	31	230
64	14	15	3	35	-	733	83	1301	-	-	-	-
81	36	26	7	54	-	-	-	-	-	-	-	-
100	73	81	7	83	-	-	-	-	-	-	-	-

Table 5: $\gamma_q = 1$, grid graph, enhanced branch-and-cut over F1.

n	$k=2$				$k=3$				$k=5$			
	Tm	T	N	LP	Tm	T	N	LP	Tm	T	N	LP
25	0.4	0.3	1	5	1.2	1.1	5	11	3.0	2.8	3	12
36	3.5	2.1	1	20	15	7.6	7	31	20	14	7	34
49	7.5	5.3	3	27	64	53	19	207	59	53	11	57
64	63	67	13	202	444	489	85	1068	328	295	15	266
81	202	79	9	131	-	-	-	-	-	-	-	-
100	298	131	11	149	-	-	-	-	-	-	-	-

Table 6: $\gamma_q = 1$, random planar graph, enhanced branch-and-cut over F1.