Control
and Cybernetics
VOL. 15 (1986) No. 3-4

Constraint generation for the graph partitioning problem

by

STANISŁAW WALUKIEWICZ

Systems Research Institute
Polish Academy of Sciences
Newelska 6
01-447 Warszawa, Poland

MARIO LUCERTINI

University of Rome
Dpt. of Informatics and System Science
Via Eudossiana, 18
00181 Rome, Italy

We study the linear programming relaxation of the graph partitioning problem and give some bounds for dual variables. The Lagrangean relaxations of the problem are also discused. Next we show how additional constraints for the problem can be generated and describe a general branch-and-bound algorithm, which includes the problem preprocessing procedure and the constraint generation procedure.

The problem of finding the last cost partition of a given weighted graph into a number of suitably defined subgraphs apears in many diverse fields as e.g. allocation of tasks in distributed computer systems, clustering analysis, taxonomy and districting, placement of components of an electronic circuit on boards. The cost of partition is defined as the total cost of edges connecting different subgraphs (see e.g. Chistofides and Brooker [2], Perng-yi et al. [11] and Simeone [12]).

As the problem is NP-complete (see e.g. Bertolazzi et al. [1]), then a general approach to solving such problems is a branch-and-bound method, which, besides suppling on optimal or near-optimal solutions, gives the possibility of taking into account additional requirements that can be expressed as integer programming constraints.

The aim of this paper is to show that additional constraints for the graph partitioning problem can be easily generated in the way described by Crowder et al. [3]. In fact we show that from a given minimal cover one has **at the same time** up to p additional constraints, where p is the number of subgraphs. Having in mind a sucess-

ful application of the constraint generation procedure described in [3], it is reasonable to assume that incorporating such a procedure in a general branch-and-bound algorithm for the graph partitioning will substantially increase its efficiency.

In contrast to the so far published papers on the graph partitioning problem we give in Section 1 the full formulation of the problem first as a nonlinear 0–1 problem and next as a linear one. In Section 2 we study the linear programming relaxation of the problem. Some other relaxations and equivalence are discussed in Section 3. In Section 4 we give a self-contained discription of a constraint generation procedure. In Section 5 we describe how such a procedure can be incorporated in a general branch-and-bound algorithm and also discuss some possibilities of preprocessing of the graph partitioning problem. Some topics fur further research are discussed in Section 6.

1. Formulation and reformulation of the problem

Consider a finite undirected graph $G=(V, E)$ with V being the set of nodes and E — set of edges, where $|V|=n\geqslant 2$ and $|E|=m>1$. Let a_i be the "weight" of the vertex i, $i=1, ..., n$ and let c_{ij} be the "capacity" of the edge $(i, j)\in E$. We assume that $c_{ij}>0$, iff $(i, j)\in E$. Without loss of generality we may assume that G has no isolated nodes.

The graph partitioning problem is to find a partition of G into at most p subgraphs G_k, $k=1, ..., p$, $2\leqslant p\leqslant n-1$, such that the total capacity of edges connecting different subgraphs is minimal and the total weight of nodes in each subgraph is not grater than a given number b. Obviously minimization of the total capacity of intersubgraphs edges is equivalent to maximization of the total capacity of edges within each subgraph $G_k=(V_k, E_k)$, therefore introducing the decision variables

$$x_{ik}=\begin{cases}1, & \text{if } i\in V_k\\ 0, & \text{if } i\notin V_k\end{cases} \qquad i=1, ..., n, \quad k=1, ..., p$$

the **graph partitioning problem** can be formulated as

$$\text{(GP)} \qquad v(GP)=\max \sum_{k=1}^{p}\sum_{i=1}^{n-1}\sum_{j>i} c_{ij}\, x_{ik}\, x_{jk} \tag{1}$$

subject to

$$\sum_{i=1}^{n} a_i\, x_{ik}\leqslant b, \qquad k=1, ..., p \tag{2}$$

$$\sum_{k=1}^{p} x_{ik}=1, \qquad i=1, ..., n \tag{3}$$

$$x_{ik}=0 \text{ or } 1, \quad i=1, ..., n, \quad k=1, ..., p. \tag{4}$$

For given problem P by $v(P)$ we denote the value of the objective function at an optimal point and by $F(P)$ $(F^*(P))$ we denote the set of all feasible (optimal) solutions to P.

Adding all constraints (2) by (3) we have that, if

$$\sum_{i=1}^{n} a_i > bp,$$

then $F(GP)=\emptyset$. Without loss of generality we may assume that all data in GP are positive integers. By our assumption in (1) we have only mp quadratic terms. We note that in practice p is small in comparison with n and m; typically $p \leqslant 4$, while $n \leqslant 100$, $m \leqslant 500$.

The problem can be linearized at least in two ways. First, if we introduce the binary variables $z_{ijk}=x_{ik}\,x_{jk}$, $i=1, ..., n-1$, $j>i$, $k=1, ..., p$ and add to GP constraints

$$2z_{ijk}-x_{ik}-x_{jk} \leqslant 0, \quad i=1, ..., n-1, \; j>i, \quad k=1, ..., p, \tag{5}$$

then we obtain the linear binary problem having $(mp+n+p)$ constraints and $p(m+n)$ variables. It is easy to check the equivalence of these two formulations.

Let $V(i)$ be the set of all nodes adjacent to a given node i, $i=1, ..., n$. We note that the variable x_{ik} appears only in the quadratic terms $x_{ik}\,x_{jk}$ when $j \in V(i)$. The second way of linearization (Goldman [7]) of GP consists in introducing bounded continuous variables $y_{ijk}=x_{ik}\,x_{jk}$, $i=1, ..., n-1$, $j>i$, $k=1, ..., p$ such that $0 \leqslant y_{ijk} \leqslant 1$ and in adding instead of (5) constraints

$$\sum_{j \in V(i)} y_{ijk} - |V(i)|\, x_{ik} \leqslant 0, \quad i=1, ..., n-1, \quad k=1, ..., p. \tag{6}$$

Therefore the linear form of the graph partitioning problem (we will denote it by the same symbols GP) is the following mixed integer programming problem

(GP)
$$v(GP)=\max \sum_{k=1}^{p} \sum_{i=1}^{n-1} \sum_{j>i} c_{ij}\, y_{ijk} \tag{7}$$

subject to

$$\sum_{i=1}^{n} a_i\, x_{ik} \leqslant b, \quad k=1, ..., p \tag{8}$$

$$\sum_{k=1}^{p} x_{ik}=1, \quad i=1, ..., n \tag{9}$$

$$\sum_{j \in V(i)} y_{ijk} - |V(i)|\, x_{ik} \leqslant 0, \quad i=1, ..., n, \quad k=1, ..., p \tag{10}$$

$$x_{ik}=0 \quad \text{or} \quad 1, \quad i=1, ..., n, \quad k=1, ..., p \tag{11}$$

$$0 \leqslant y_{ijk} \leqslant 1, \quad i=1, ..., n, j>i, \quad k=1, ..., p \tag{12}$$

Such a linear form has fewer constraints, namely $n+p(n+1)$, and the same number of variables, but among them mp are continuous ones. We also note that the constraint matrix for (8)–(10) is sparse; its density can be estimated as

$$d=\frac{2mp+3np}{(n+p+np)(mp+np)}\leqslant\frac{3}{n+p+np}$$

So e.g. for $n=100$ and $p=4$ we have $d\leqslant 0.6\%$. The sparsity of the matrix is an important factor of a sucsessful application of a constraint generation procedure.

2. Linear programming relaxation

For a given integer programming problem P by $\bar{P}$ we will denote its linear programming relaxation. So $\overline{GP}$ differs from GP by the constraints (11) as instead of them we have

$$0\leqslant x_{ik}\leqslant 1, \quad i=1,\ldots,n, \quad k=1,\ldots,p. \tag{13}$$

The linear programming problem (7)—(10), (12) and (13) is called the **linear graph partitioning problem.**

If we define the dual variables as u_k, v_i, w_{ik}, q_{ijk}, t_{ik} respectively to the constraints (8), (9), (10), (12) and (13), then the dual to $\overline{GP}$ takes the form

$$(\overline{DGP})\ v(\overline{DGP})=\min\Big[b\sum_{k=1}^{p}u_k+\sum_{i=1}^{n}v_i+\sum_{k=1}^{p}\sum_{i=1}^{n}\sum_{j>i}q_{ijk}+\sum_{k=1}^{p}\sum_{i=1}^{n}t_{ik}\Big] \tag{14}$$

subject to

$$a_i u_k+v_i-|V(i)|w_{ik}+t_{ik}\geqslant 0, \quad i=1,\ldots,n, \quad k=1,\ldots,p \tag{15}$$

$$w_{ik}+w_{jk}+q_{ijk}\geqslant c_{ij}, \quad i=1,\ldots,n-1,\ j>i, \quad k=1,\ldots,p \tag{16}$$

$$u_k\geqslant 0,\ w_{ik}\geqslant 0,\ q_{ijk}\geqslant 0,\ t_{ik}\geqslant 0, \quad i=1,\ldots,n-1,\ j>i, \quad k=1,\ldots,p \tag{17}$$

$$v_i \text{ unrestricted in sign}, \quad i=1,\ldots,n \tag{18}$$

From (15) one has

$$v_i\geqslant|V(i)|w_{ik}-a_i u_k-t_{ik} \quad \text{for} \quad k=1,\ldots,p. \tag{19}$$

Therefore

$$v_i=\max_{k=1,\ldots,p}\{|V(i)|w_{ik}-a_i u_k-t_{ik}\}, \quad i=1,\ldots,n \tag{20}$$

As the variables w_{ik} do not appear in the objective function of $\overline{DGP}$, but q_{ijk} do, then by (16) and (17) we have that:

$$\text{if } w_{ik}+w_{jk}\geqslant c_{ij}, \quad \text{then } q_{ijk}=0,$$

$$\text{if } w_{ik}+w_{jk}<c_{ij}, \quad \text{then } q_{ijk}=c_{ij}-(w_{ik}+w_{jk}).$$

Therefore

$$q_{ijk}=\max\{0, c_{ij}-(w_{ik}+w_{jk})\}, \quad i=1, ..., n=1, j>i, \quad k=1, ..., p \tag{21}$$

If $w_{ik}>0$, then by the complementary slackness we have equality in (10), but this means that if $x_{ik}=1$, then $y_{ijk}=1$ for all $j\in V(i)$. Hence we are interested in having w_{ik} as large as possible. But w_{ik} is bounded from above by (15)

$$w_{ik}\leqslant\frac{a_i u_k+v_i+t_{ik}}{|V(i)|}, \quad i=1, ..., n, \quad k=1, ..., p \tag{22}$$

Since graph G has not isolated nodes $|V(i)|>0$ for all i, therefore (21) can be rewritten as

$$q_{ijk}=\max\left\{0, c_{ij}-\left(\frac{a_i u_k+v_i+t_{ik}}{|V(i)|}+\frac{a_j u_k+v_j+t_{jk}}{|V(j)|}\right)\right\} \tag{23}$$

for $i=1, ..., n-1, \quad j>i, \quad k=1, ..., p.$

3. Other relaxations of GP

In this section we study some relaxation of *GP* which seem to be promising from the computational point of view.

Consider again *GP* in its linear form (7)–(12). If we define $h_{ik}\geqslant 0$, $i=1, ..., n$, $k=1, ..., p$, to be Lagrangean multipliers for (10), then a **Lagrangean relaxation** L of *GP* is a mixed integer programming problem of the form

$$(L) \quad v(L)=\max\left[\sum_{k=1}^{p}\sum_{i=1}^{n}\sum_{j>1} c_{ij} y_{ijk}-\sum_{k=1}^{p}\sum_{i=1}^{n} h_{ik}\left(\sum_{j\in V(i)} y_{ijk}+\right.\right.$$
$$\left.\left.-|V(i)|x_{ik}\right)\right] \tag{24}$$

Subject to (8), (9), (11) and (12). We note that continuous variables do not appear in the constraints of L. If $(\bar{x}, \bar{y})$ is an optimal solution to L, then for given h_{ik}

$$\bar{y}_{ijk}=\begin{cases}1, & \text{if } c_{ij}-h_{ik}>0\\ 0, & \text{if } c_{ij}-h_{ik}<0\\ \alpha, & \text{where } 0\leqslant\alpha\leqslant 1, \quad \text{if } c_{ij}-h_{ik}=0\end{cases}$$

So optimal values of the continuous variables can be easily computed.

A reasonable choice of h_{ik} is putting $h_{ik}=\bar{w}_{ik}$ at the beginning, where $\bar{w}_{ik}$ is a part of an optimal solution to $\overline{DGP}$. In the following iterations we compute h_{ik} using subgradient technique (see e.g. Geoffrion [5]).

In general problem L has not the Integrality Property [5], therefore it can provide better upper bound on $v(GP)$ than the linear programming relaxation.

Next relaxation of *GP* we obtain from L by aggregating all constraints (8) into one of the form

$$\sum_{k=1}^{p}\sum_{i=1}^{n} g_k a_i x_{ik}\leqslant b\sum_{k=1}^{p} g_k, \tag{25}$$

where $g_k \geqslant 0$, $k=1, \ldots, p$ are given aggregation coefficients. Without loss of generality we may assume that

$$\sum_{k=1}^{p} g_k = 1$$

In such a way we obtain a **surrogate Lagrangean relaxation** of *GP*, which can be formulated as

$$(SL) \quad v(SL) = \max \Big[\sum_{k=1}^{p} \sum_{i=1}^{n} \sum_{j>1} c_{ij}\, y_{ijk} - \sum_{k=1}^{p} \sum_{i=1}^{n} h_{ik} \Big(\sum_{j \in V(i)} y_{ijk} + - |V(i)|\, x_{ik} \Big) \Big] \tag{26}$$

subject to

$$\sum_{k=1}^{p} \sum_{i=1}^{n} g_k\, a_i\, x_{ik} \leqslant b \tag{27}$$

$$\sum_{k=1}^{p} x_{ik} = 1, \quad i=1, \ldots, n \tag{28}$$

$$x_{ik} = 0 \quad \text{or} \quad 1, \quad i=1, \ldots, n, \quad k=1, \ldots, p \tag{29}$$

$$0 \leqslant y_{ijk} \leqslant 1 \quad i=1, \ldots, n, \; j \; i, \quad k=1, \ldots, p \tag{30}$$

This is in fact the multiple-choice knapsack problem and for such a problem we have efficient in practice methods for solving it or its linear programming relaxation (see Dudziński and Walukiewicz [4] for a survey). We also note that it is possible to reduce the number of binary variables in *SL* using the reduction methods for multiple-choice knapsack problem [4]. We will discuss this question in Section 6.

We can propose two methods of computing aggregation coefficients: First start with $\bar{u}_k$, where $\bar{u}_k$ is a part of an optimal solution to $\overline{DGP}$ and normalize it. In the subsequent iterations the value of g_k may be changed in ways described in Karwan and Rardin [8].

The second methods bases on the aggregation of equalities in bounded integer programming problems (see Onyekwelu [9]). Although (8) are inequalities, but they are disjoint and in an optimal solution many of them are equalities. So if for a moment we will treat (8) as equalities, then we can multiply them by an integer aggregation coefficients $g_k \geqslant 0$, which are e.g. first p prime numbers i.e. $g_1=1$, $g_2=2$, $g_3=3$, $g_4=5$ etc. In such a way we obtain multiple-choice knapsack problem *SL*. It is resonable to assume that there is a good chance that $v(SL)=v(L)$, i.e. such coefficients are really aggregation coefficients. If it is not the case then we may correct the vector $g \in Z^p$, where Z^p is a set of all p-dimensional vectors with nonnegative integer components, in a way described by Walukiewicz in [13].

We finish this section with the other equivalent formulation of the graph partitioning problem. Consider again the formulation (1)–(4). Introducing p integer variables $z_k \geqslant 0$, $k=1, ..., p$, we may write (2) as equalities

$$\sum_{i=1}^{n} a_i x_{ik} + z_k = b, \quad k=1, ..., p$$

and next aggregate them into

$$\sum_{k=1}^{p} g_k \left(\sum_{i=1}^{n} a_i x_{ik} + z_k \right) = b \sum_{k=1}^{p} g_k,$$

where g_k is the k-th prime number. Thus the graph partitioning problem (1)–(4) is equivalent to

$$(GP) \qquad v(GP) = \sum_{k=1}^{p} \sum_{i=1}^{n} \sum_{j>i} c_{ij} x_{ik} x_{jk}$$

subject to

$$\sum_{k=1}^{p} g_k \left(\sum_{n=1}^{n} a_i x_{ik} + z_k \right) = b \sum_{k=1}^{p} g_k$$

$$\sum_{k=1}^{p} x_{ik} = 1, \quad i=1, ..., n$$

$$x_{ik} = 0 \text{ or } 1, \quad i=1, ..., n, \quad k=1, ..., p$$

This is an equality — constrainted quadratic knapsack problem with multiple-choice constraints.

4. Constraint generation

The aim of this section is to provide self-contained discription of the constraint generation procedure.

The convex hull of 0–1 solutions to the single inequality

$$\sum_{j=1}^{n} a_j x_j \leqslant b \tag{31}$$

is called the **knapsack polytope** W, so

$$W = \operatorname{conv} \left\{ x \in R^n : \sum_{j=1}^{n} a_j x_j \leqslant b, \quad x_j \in \{0, 1\} \right\}$$

The **dimension** of W, dim W, is the smallest dimension of the real space containing W. Without loss of generality we may assume that dim $W=n$. An equality $hx=h_0$ in R^n is a **supporting hyperplane** of W, if $hx \leqslant h_0$ for any $x \in W$ and $H=W \cap \{x \in R^n:$

$: hx=h_0\} \neq \emptyset$. The polytope H and the corresponding inequality $hx \leqslant h_0$ is called a **face** of W. If dim H is $n-1$, then H is called a **facet** of W. A complete list of facets for the knapsack polytope is unknown to date (see Padberg [10]), but some facets can be obtained from minimal covers defind below.

A subset S of $N=\{1, ..., n\}$ is called a minimal cover of (31) if

$$\sum_{j \in S} a_j > b \quad \text{and} \quad \sum_{j \in S} a_j - a_k \leqslant b \quad \text{for all } k \in S.$$

Then every binary solution x to (31) satisfies

$$\sum_{j \in S} x_j \leqslant |S| - 1. \tag{32}$$

Usually $|S|<n$ and (32) defines only a face of W. To obtain a facet, i.e. the best possible inequality defining W, (32) must be lifted by so called **lifting procedure.** Initially we set $h_j=1$ for $j \in S$ and $h_0=|S|-1$. For every $k \in N-S$ we compute

$$v_k = \max \left\{ \sum_{j \in S} h_j x_j;\ \sum_{j \in S} h_j x_j \leqslant h_0 - a_k,\ x_j \in \{0, 1\}, \quad j \in S \right\}, \tag{33}$$

define $h_k=h_0-v_k$, redefine S to $S-\{k\}$ and repeat until $N-S$ is empty. The resulting inequality is a facet of W.

Consider now a general 0–1 programming problem formulated as

$$(P) \qquad v(P) = \max \{cx: Ax \leqslant b,\ x_j \in \{0, 1\}, \quad j=1, ..., r\},$$

where A is m by r matrix. Let $Q=\text{conv } F(P)$ and let W_i be the knapsack polytope corresponding to the i-th constraint. Clearly we have

$$Q \subseteq \bigcap_{i=1}^{m} W_i \tag{34}$$

Equality in (34) does not, in general, hold, but does hold if, for example, P decomposes into m knapsack problems. Therefore if matrix A is sparse then it is reasonable to expect that the intersection of m knapsack polytopes W_i provides a fairly good approximation to Q. Computational results [3] confirm that it is a reasonable assumption.

The idea of so called **constraint generation approach** consist in solving $\bar{P}$ and if $\bar{x} \in F^*(\bar{P})$ is not a binary vector, then we look for a minimal cover inequality (32) which cuts off $\bar{x}$, if such an inequality exists. To do this we consider a single inequality $a^i x \leqslant b_i$ of P and using a variable substitution $x'=1-x_j$, where necessary, we bring it into a form of (31) with all nonzero coefficients positive. Dropping the index i for notational convenience we consider thus a knapsack inequality

$$\sum_{j \in N} a_j x_j \leqslant b, \tag{35}$$

where $N \subseteq \{1, ..., r\}$.

For given $\bar{x} \in F^*(\bar{P})$ and (35) we consider the following knapsack problem

$$q = \min \left\{ \sum_{j \in N} (1-\bar{x}_j) y_j:\ \sum_{j \in N} a_j y_j > b,\ y_j \{0, 1\}, \quad j \in N \right\}. \tag{36}$$

It can be proved that there exists a minimal cover inequality (32) which cuts off $\bar{x}$ if and only if $q<1$ (see [3]). Such a minimal cover S^* is defined in the following way: Let y^* be an optimal solution to (36). If y^* is unique, then $S^*=\{j\in N: y^*=1\}$. If y^* is not unique, then among the optimal solutions there exists at least one solution for which the corresponding set S^* is a minimal cover for (35).

We note that the additional constraint $hx\leqslant h_0$ preserves the sparsity of the matrix A as we have $h_j\neq 0$ only if $a_j\neq 0$ in (35). This is a very important difference between additional constraints and traditional cutting planes, as the last ones are typically dense and lead to explosive storage requirements and numerical instability. Computational results reported in [3] show that relatively few additional constraints added to P increase the efficiency of the branch-and-bound method substantially.

Moreover instead of solving some number of knapsack problems (36) and (34) we can solve its linear programming relaxations. It is well-known that the linear knapsack problem can be solved in $0(n)$ time [4]. The inequality $hx\leqslant h_0$ obtained in such a way is not, in general, a facet of a corresponding knapsack polytope, but as [3] shows, still has a big computational value.

A generalization of a minimal cover is so called **(1, k)-configuration.** Consider again (35) and let $T\subseteq N$ and $t\in N-T$. If

$$\sum_{j\in T} a_j\leqslant b \text{ and } T'\cup\{t\} \text{ is a minimal cover for (35) for every } T'\subseteq T \text{ such that } |T'|=k, \qquad (37)$$

then the set $T\cup\{t\}$ is called a $(1, k)$-configuration for (35). It is easy to see that every zero-one solution to (35) satisfies the following set of inequalities:

$$(s-k+1)\,x_t+\sum_{j\in T(r)} x_j\leqslant s, \qquad (38)$$

where $T(s)\subset T$ varies over all subsets of cardinality s of T and $k\leqslant s\leqslant |T|$. If $k=|T|$, then the $(1, k)$-configuration is a minimal cover. If $T\cup\{t\}\neq N$, then an inquality (38) can be transformed to a facet of the knapsack polytope by means of the lifting procedure (33).

5. Branch-and-bound methods

We now describe how a general branch-and-bound method (see e.g. Geoffrion and Marsten [6]) can be modified to solve GP in the linear form (7)–(12) of the graph partitioning problem.

First we solve $\overline{GP}$ obtaining $(\bar{x}, \bar{y})\in F^*(\overline{GP})$. If $\bar{x}$ is a binary vector, than we stop — GP was solved. If $\bar{x}$ is not a binary vector then we solve GP in three phases:

1. Preprocessing Phase,
2. Constraints Generation Phase,
3. Branch-and-Bound Phase.

1. Preprocessing Phase

Let $v(GP)$ be a lower bound on the value of the optimal solution to GP, i.e. $v(GP)$ may be taken from practice or obtained from so far done computations. Let d_{ik} be the optimal reduced cost in the simplex tableau for $\overline{GP}$. Then for any nonbasic variable x_{ik} the following holds:

i) If $\bar{x}_{ik}=0$ and $v(\overline{GP})+d_{ik}<v(GP)$, then $x^*_{ik}=0$ in every optimal solution (x^*, y^*) to GP.

ii) If $\bar{x}_{ik}=1$ and $v(\overline{GP})-d_{ik}<v(GP)$, then $x^*_{ik}=1$ in every optimal solution (x^*, y^*) to GP.

In the branch-and-bound method the above test is made each time when a new, better, feasible solution is obtained. If the test is successful, then x_{ik} is removed, number of zero-one variables is decreased by one and moreover further reduction of variables and/or constraints are possible:

i') If $x^*_{ik}=0$, then the i-th constraint (9) reads

$$\sum_{\substack{t=1\\ r\neq k}}^{p} x_{ir}=1$$

and from (10) follows that $y^*_{ijk}=0$ for all $j\in V(i)$ and therefore we may remove these y_{ijk} from GP, wich in turn may give further reductions.

ii') If $x^*_{ik}=1$, then by (9) $x^*_{ir}=0$ for $r=1, \ldots, p$, $r\neq k$ and further reduction of y_{ijk} and/or x_{ik} is possible.

The above reduction for a given $v(GP)$ ends when all nonbasic variables x_{ik} passed the tests i) and ii). Having in mind that the similar procedure in the case of the multiple-choice knapsack problem reduces up to 90% of variables, one should expect similar results for GP.

As a result of preprocessing we obtain, in general, GP with fewer variables and constraints, but for simplicity of notation, we will not change the bounds for the parameters in (7)–(12) and we will assume that in a present optimal solution $(\bar{x}, \bar{y})\in \in F^*(\overline{GP})$, the vector $\bar{x}$ is not a binary one. Then we proceed to the

2. Constraint Generation Phase

Only constraints (8) can produce additional constraints in the way described in Section 4. Moreover, as the coefficients in all constraints are the same then a given minimal cover or $(1, k)$–configuration gives at the same time up to p different additional constraints, where p is the number of subgraphs. So for constraint generation we can drop the index k in x_{ik} and write (8) as a single constraint (35)

$$\sum_{j\in N} a_j x_j \leqslant b \tag{39}$$

where $N=\{1, \ldots, n\}$. Obviously $a_j>0$ for $j\in N$.

Let $(\bar{x}, \bar{y})\in F^*(\overline{GP})$. We define $\hat{x}=(\hat{x}_1, \ldots, \hat{x}_n)$ as

$$\hat{x}_i=\min_{k=1,\ldots,p} \bar{x}_{ik}$$

and $N_1=\{j\in N: \hat{x}_j>0\}$. Now we solve the knapsack problem (36) with N replaced by N_1 and $\bar{x}_j$ replaced by $\hat{x}_j$. Obviously $|N_1|\leqslant|N|$ and if $|N_1|$ is still a large number, than instead of solving (36) we solve its linear programming relaxation putting $0\leqslant y_j\leqslant 1$ for $j\in N_1$. Let $\bar{y}$ be an optimal solution to the above problem; then we check if the set $S'=\{j\in N: \bar{y}_j>0\}$ is a minimal cover S for (39). If not we obtain a minimal cover from S' by dropping some of the variables. In this way we obtain the inequality (32).

If $N_1-S\neq\emptyset$ then we extend (32) by the lifting procedure (33). Also here we may solve the linear knapsack problem instead (33) by setting $0\leqslant x_j\leqslant 1$ for $j\in N_1-S$. Let $\bar{v}_k$ be the value of such a problem, for $k\in N_1-S$. Then we have $h'_k=h_0-\lfloor v_k\rfloor\leqslant h_k$, where $\lfloor a\rfloor$ means the integer part of the number a. So by solving linear knapsack problems we underestimate the "true" extension coefficients in $hx\leqslant h_0$.

As we have solved (36) and (33) in near-optimal way, then we have to check, if $hx\leqslant h_0$ cuts off $\bar{x}$. If it is not the case then we try to find $(1, k)$–configuration and the corresponding inequality (38). If (38) after the extension do not cut off $\bar{x}$ than the trial to construct an additional constraint in a near-optimal way was unsuccessful and we may go to the Branch-and-Bound Phase or solve (33) and (36) in the optimal way. If the trial is successful then we lift $hx\leqslant h_0$ for $j\in N-N_1$.

3. Branch-and-Bound Phase

Obviously *GP* can be solved by any branch-and-bound algorithm, but few remarks are in order.

First, branching should be done in such a way that two subproblems have the same structure, i.e. we have to take into account the SOS constraints (10) (see [6]).

Second, some balance between preprocessing, constraints generating and branch--and-bound phases has to be kept in an efficient algorithm for *GP*. At present we think that we have to use the full possibility of preprocessing, then of constraints generation and at the end the full possibility of branch-and-bound. So we try to avoid branching as much as possible.

Third, it is advisable to have at least two algorithms for solving a linear knapsack problem: one for ordered variables and the second one without ordering of variables [4] and apply one of them depending on how big is $|N_1|$.

6. Conclusions

In many applications of the graph partitioning problem we have $a_i=1$ for $i=1, \ldots, n$. Then each subgraph $G_k=(V_k, E_k)$ contains no more than b nodes. One can check that the Lagrangean relaxation L has the Integrality Property [5] in this case. In other words L cannot give better bounds than the linear programming relaxation as we have

$$v(L)=v(\bar{L})=v(\overline{GP}).$$

And for the surrogate Lagrangean relaxation SL we have

$$v(SL) \geqslant v(L).$$

But even in this case solving L or SL will, in general require less time then solving $\overline{GP}$, for instance we solve $\overline{GP}$ only once at the beginning of the branch-and-bound and next for subproblems solve corresponding surrogate Lagrangean relaxations using efficient methods described in [4].

Next particular cases come from the requirement that the connected graph $G=(V, E)$ should be partitioned into at most p connected subgraphs $G_k=(V_k, E_k)$. In the paper in preparation we show that the connected requirements can be expressed in the form of linear constraints and such an approach gives a new algorithm for solving some particular cases of the graph partitioning problem.

Finally, we stress the importance of study of the quadratic multiple-choice knapsack problem. It is reasonable to assume that many results known for the (linear) multiple-choice knapsack problem [4] can be modified for the case of quadratic multiple-choice knapsack problem.

References

[1] BERTOLAZZI P., LUCERTINI M., MARCHETTI SPACCAMELA A. Analysis of a class of graph partitioning problems. *RAIRO Informatique Theorique*, **16** (1982), 255–261.

[2] CHRISTOFIDES N., BROOKER P. The optimal partitioning of graphs. *SIAM J. App. Math.* **30** (1976), 55–69.

[3] CROWDER H., JOHNSON E. L., PADBERG M. W. Solving large-scale zero-one linear programming problems, *Opns. Res.* **31** (1983), 803–834.

[4] DUDZIŃSKI K., WALUKIEWICZ S. Knapsack problem and its generalizations. Report of Systems Research Institute, Warsaw 1984.

[5] GEOFFRION A. M. Lagrangean relaxation for integer programming. *Math. Prog. Study*, **2** (1974), 82–114.

[6] GEOFFRION A. M., MARSTEN R. E. Integer programming algorithms: a framework and state-of-the-art-survey. *Management Sci*, **18** (1972), 465–491.

[7] GOLDMAN A. J. Linearization in 0–1 variables: A clarification. *Opns. Res.* **31** (1983), 946–948.

[8] KARWAN M. H., RARDIN R. L. Surrogate dual multiplier search procedures in integer programming. *Opns. Res.* **32** (1984), 52–69.

[9] ONYEKWELU D. C. Computational viability of a constraint aggregation scheme for integer programming problems. *Opns. Res.* **31** (1983), 795–802.

[10] PADBERG M. W. Covering, packing and knapsack problems. *Annals of Discrete Math.* **5** (1979), 139–183.

[11] PERNG-YI R., LEE E. Y. S., TSUCHIYA M. A task allocation model for distributed computing systems. *IEEE Transactions on Computers*, **C-31** (1982), 41–47.

[12] SIMEONE B. Optimal graph partitioning. *EJOR*, (to appear).

[13] WALUKIEWICZ S. Some aspects of integer programming duality. *EJOR*, **7** (1981), 196–202.

Generowanie ograniczeń dodatkowych w zagadnieniu rozbicia grafu

Przeanalizowano zadanie programowania liniowego stowarzyszone z zagadnieniem rozbicia grafu i podano oszacowania na wartości zmiennych dualnych w punkcie optymalnym. Opisano również relaksację Lagrange'a dla tego zagadnienia. Następnie pokazano jak można generować ograniczenia dodatkowe i opisano ogólny algorytm podziału i oszacowań, który zawiera procedurę wstępnego przetwarzania i procedurę generowania ograniczeń dodatkowych.

Генерирование дополнительных ограничений в задаче разбиения графа

В статье проведен анализ задачи линейного программирования, сопряженной с проблемой разбиения графа и даны оценки значений двойственных переменных в оптимальной точке. Описана также релаксация Лагранжа для этой проблемы. Затем показано, как можно генерировать дополнительные ограничения и описан общий алгоритм ветвей и границ, содержащий процедуру начальной обработки и процедуру генерирования дополнительных ограничений.