Control
and Cybernetics
VOL. 13 (1984) No. 4

Three algorithms for calculating surrogate constraints in integer programming problems

by

KOICHI MIZUKAMI

University of Hiroshima
Faculty of Integrated Arts and Sciences
Hiroshima, Japan

JAROSŁAW SIKORSKI*

Polish Academy of Sciences
Systems Research Institute
Warszawa, Poland

The surrogate dual program offers a more effective bound than the Lagrangean dual on the primal optimal value in branch-and-bound methods since it yields usually a tighter duality gap.

Three algorithms for solving the surrogate dual program are presented and compared experimentally. Two of them are based on a quasi-subgradient search method. One has been presented in [3], while another is a new proposal. Both are endowed with a new effective stop test. The remaining one originates from generalised programming and follows the scheme discussed in [3]. All of them require solution of a knapsack problem in each iteration.

1. Introduction

The surrogate constraint, as a single constraint which can substitute for a whole set of constraints, was introduced into mathematical programming by Glover [5], for binary integer-programming problems. Since Luenberger [10] used a similar approach in duality for quasiconvex programs a more theoretical stream of work was initiated, e.g. [7], [6], [8]. On the other hand, surrogate constraints have demonstrated their utility as a part of branch-and-bound methods in integer programming, e.g. [1], [9].

Two methods for calculating surrogate constraints are described in this paper from a more practical than theoretical point of view. They are discussed and compared in their algorithmic forms suitable for implementation. Obviously some parts of the presented algorithms can take different forms while being still based on the same method, so it is not claimed here that these representations are unique or even

*) Visiting research associate at the University of Hiroshima on leave from from Polish Academy of Sciences in the period between October 1982 and March 1984.

best. They are described as examples in order to show how each of these two methods proceeds and what its most important features are. Finally a comparison is done by a numerical experiment using integer test problems with a nonlinear objective function and linear constraints.

Briefly, the first method consists in constructing a sequence of nested sets such that the "volume" tends to zero and unsatisfactory points are rejected in each step. It will be called the decaying polytope algorithm, and is based on the first scheme proposed by Dyer [3]. The other method is an application of a search direction scheme with an a priori stated step size sequence. A linear functional strongly supporting the interior of the upper level set of a certain function is used for construction of a search direction. Two algorithms were developed using such vectors, named quasisubgradients [7], which define such a functional. The quasi-subgradient Algorithm B is based on the second of Dyer's schemes [3] but is endowed with a new effective stop test. Algorithm C is a new proposal which exploits the same ideas as Algorithm B.

In the next section of the paper only those parts of the general theory appear which are necessary for the presentation of the algorithms.

2. Preliminaries

Consider the following integer program

$$(P) \qquad \hat{f}=\min \{f(x): x\in X,\ g(x)\leqq 0,\ x \text{ integer}\},$$

where $f: R^n\to R$, $g: R^n\to R^m$ and $X=\{x\in R^n: 0\leqq x_k\leqq u,\ k=1, \dots, n\}$.

It is not necessary heve to assume lower semicontinuity of f and continuity of g in view of the fact that the x's are integer although the results cited hereafter were proved in a more general case under these assumptions.

When f is separable i.e. can be represented as a sum $\sum_{k=1}^{n} f_k(x_k)$, and constraints are linear: $g(x)=Ax-b$, where matrix A and vector b are both integer, the problem is often called the multidimensional knapsack problem (nonlinear if f_k are nonlinear). All test problems in the numerical experiments are exactly of this form but these particular assumptions are not necessary for the presentation of all algorithms. Thus for (P) with only the initial assumptions, for any $w\in R^m$, $w\geqq 0$ the following relaxation is defined [6].

$$(S) \qquad h(w)=\min \{f(x): x\in X(w),\ x \text{ integer}\},$$

where $X(w)=\{x\in X: (w, g(x))\leqq 0\}$. Then

$$(SD) \qquad \hat{h}=\sup \{h(w): w\in R^m,\ w\geqq 0\}$$

is the surrogate dual program for (P). $h: R^m\to R$ is called the surrogate dual function and the constraint defining $X(w)$ a surrogate constraint. Program (SD) is aimed

at finding the best surrogate constraint, if it exists. The two properties of the set $X(w)$: $X(0)=X$ and $X(sw)=X(w)$ for $s>0$, yield respectively two features of the surrogate dual function : $h(0)\leqq h(w)$ for all $w\geqq 0$ and $h(sw)=h(w)$. Thus zero may be excluded from the feasible set and an arbitrary normalization of the vectors w is possible, which allows us to restrict the feasible set in (SD) to a compact one.

The following results state the basic properties of the problem.

PROPOSITION *1* [7] (Weak Duality Theorem) $\hat{h}\leqq\hat{f}$

PROPOSITION *2* [3] $\hat{h}=\hat{f}$ and $\hat{h}$ is achieved if and only if $h(w)=f(x)$ for some $w\geqq 0$ and x feasible in (P). Then x solves (P) and w solves (SD).
If the equality does not hold the quantity $(\hat{f}-\hat{h})$ is called the duality gap. The smaller the duality gap, the more valuable is the problem (SD).

PROPOSITION *3* [7] $h(w)$ is a lower semicontinuous, quasiconcave function on $R^m_+=\{w\in R^m: w\geqq 0\}$.

PROPOSITION *4* [3] $\hat{h}$ is attained on a relatively open convex cone contained in R^m_+.
As a result of the last proposition "sup" in the statement of (SD) can be replaced by "max". It should be stressed however that this is possible only because all x belong to the integer lattice. Then the feasible set in (P) has no cluster point; if it has, $\hat{h}$ may not be equal to any $h(w)$ (see example 1 in [3]). Proposition 3 indicates that (SD) is rather a difficult problem from the operative point of view for it involves maximizing a function which can be discontinuous. Thus some additional characterization of the value $\hat{h}$ can be helpful. In order to achieve a more geometrical characterization, a concise notation for certain families of sets is convenient. Let $T(\alpha)=\{x\in X: f(x)\leqq\alpha, x \text{ integer}\}$, for every $\alpha\in R$, be a lower level set of the function f. Then the inequality $h(w)\leqq\alpha$ is equivalent to the condition $T(\alpha)\cap X(w)\neq\emptyset$. Denote the optimal set of (S), i.e. $\{x\in X(w): h(w)=f(x), x \text{ integer}\}$, by $Q(w)$. Then clearly the set $Q(w)$ can be expressed as $T\big(h(w)\big)\cap X(w)$. Let $R(\alpha)=g\big(T(\alpha)\big)$ denote the image of a level set under the constraint function g. Also denote by $R^*(\alpha)$ the set $\{w\in R^m_+: (w,v)\geqq 0 \text{ for all } v\in R(\alpha)\}$.

PROPOSITION *5* [3] The value $\hat{h}$ can be determined by the program
$\min\{\alpha: \text{int } R^*(\alpha)=\emptyset\}$.
Plainly $0\in\text{co } R(\alpha)$ implies $\text{int } R^*(\alpha)=\emptyset$ where "co" means convex hull and "int" — interior relative to R^m_+.

PROPOSITION *6* [3] $\text{int } R^*(\alpha)=\emptyset$ if and only if $\text{co } R(\alpha)\cap R^m_-\neq\emptyset$ $(R^m_-=-R^m_+)$

COROLLARY *1. The value $\hat{h}$ can be determined by the program*
$\min\{\alpha: co\ R(\alpha)\cap R^m_-\neq\emptyset\}$.
Since $\hat{h}$ is attained for some w obviously

COROLLARY 2. *$h(w)=\hat{h}$ if and only if co $R(h(w)) \cap R^m_- \neq \emptyset$.*

The first characterization (Proposition 5) is essential in the decaying polytope algorithm while the result of Corollary 1 serves as a reasonable stop test in both quasi-subgradient algotrihms.

3. Decaying polytope algorithm

The algorithm described in this section uses the arbitrary normalisation: $||w||=1$. The L_1 norm is chosen: $||w||=\sum_{l=1}^{m} |w_l|$. Hence (SD) can be rewritten in the form

$$\text{(SD')} \qquad \hat{h}=\max \{h(w): w \in R^m_+, \sum_{l=1}^{m} w_l=1\}.$$

Define the feasible set $K'=\{w \in R^m_+ : (w, e)=1\}$ where $e=[1, ..., 1]$. Then K' is compact and convex.

A polytope of the form $W=\{w \in K' : (w, g^j) \geqq 0, j \in J\}$, where J is a finite index set of vectors g^j, is used as an outer approximation of $R^*(\alpha)$. The convergence of the algorithm depends on finding an interior point (relative to K') of such a polytope in each iteration. A measure of "volume" of the polytope interior is also required. Dyer [3] proposed the following linear program as a solution for both tasks.

$$\text{(L')} \qquad r=\max \eta;\ (w, g^j)-\pi_j \eta \geqq 0\ (j \in J), \quad (w, e)=1,\ w \geqq 0,$$
$$\text{where } \pi_j=\nu\left[(g^j, g^j)-(e, g^j)^2/m\right]$$

Number r will be called the radius of W and it is not difficult to see that int $W \neq \emptyset$ if and only if $r>0$. When $r>0$, the program (L') yields one interior point of W which is used to permit a wider choice depending on a scalar parameter. New vectors are successively added to the set defining W and the lower bound α on $\hat{h}$ is enlarged, whenever possible, during the calculations. Thus a sequence of polytopes and a sequence of lower bounds for $\hat{h}$ are generated.

Algorithm A

STEP 0: Set $0<\theta \leqq 1$, $w^1 \in K'$, $W_0 \leftarrow K'$, $\alpha_1 \leftarrow -\infty$, $i \leftarrow 1$.
STEP 1: Determine any $x^i \in Q(w^i)$. Let $g^i=g(x^i)$.
STEP 2: If $f(x^i)>\alpha_i$, $\alpha_i \leftarrow f(x^i)$.
STEP 3: If $g^i \leqq 0$, then stop.
STEP 4: $W^i \leftarrow W_{i-1} \cap \{w \in R^m : (w, g^i) \geqq 0\}$.
STEP 5: If the radius of W_i, $r_i \leqq 0$, then stop. Otherwise determine
$d_i=-(w^i, g^i)/[(\bar{w}^i, g^i)-(w^i, g^i)]$, where $\bar{w}^i$ is a solution of (L').
STEP 6: $w^{i+1} \leftarrow (1-c_i)\,\bar{w}^i+c_i w^i$, where $c_i=(1-d_i)(1-\theta)$.
STEP 7: $i \leftarrow i+1$. Go to Step 1.

Clearly at each iteration W_i has the form required for the statement of (L') and α_i is a lower bound for $\hat{h}$. In Step 1 the surrogate relaxation (S) is solved for a current vector w^i. If the stop test in Step 3 is satisfied, then x^i is also a solution for (P) and

the duality gap equals zero ($\alpha_i=\hat{h}=\hat{f}$). In Step 4 the new cutting plane implied by the sub-problem solution is added to the set defining W_i. If the stop test in Step 5 is satisfied, then int $W_i=\emptyset$, hence int $R^*(\alpha_i)=\emptyset$ (because $R^*(\alpha_i)\subset W_i$), which implies $\alpha_i=\hat{h}$. Figure 1 shows the geometrical interpretation of the choice of w^{i+1}.

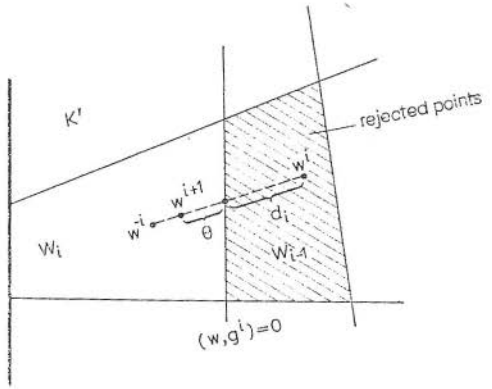

Fig. 1.

The main convergence result proved in [3] is

PROPOSITION 7. If (S) is feasible then Algorithm A terminates in Step 3 or 5 with $\alpha_i=\hat{h}$.

The result strongly depends on the fact that the feasible set in (P) is finite and the number of its elements bounds the number of iterations.

4. Quasi-subgradient algorithms

It has been proved [7] that $g(x)$ for $x\in Q(w)$ defines a linear functional strongly supporting the interior of the upper level set of h in w (if it is nonempty): $(w, g(x))<(u, g(x))$ for all u such that $h(u)>h(w)$. Any vector possessing this property was christened a quasi-subgradient as a notion corresponding to a subgradient of a concave function. Remember that h is only quasiconcave so in general a subgradient does not exist at each point of the domain. In both quasi-subgradient algorithms presented here $g(x)$ is used for the construction of a search direction and both use the L_2 normalisation: $\nu\sum_{1=l}^{m} w_l^2=1$. Hence (SD) can be stated in the form

$$\text{(SD'')} \qquad \hat{h}=\max\left\{h(w): w\in R_+^m, \quad \sum_{1=l}^{m} w_i^2=1\right\}.$$

The feasible set $K''=\{w\in R_+^m: (w, w)=1\}$ is compact and spherically convex. A convex hull of the form $C=\operatorname{co}\{g^j: j\in J\}=\{y\in R^m: y=\sum_{j\in J}\beta_j g^j, (\beta, e)=1, \beta\geqq 0\}$ is used as an inner approximation of co $R(\alpha)$. Although the convergence of the algorithms is independent of C the measurement of a distance between C and R_-^m serves as a rea-

sonable stop test. The following linear program is proposed as a way to measure such distance.

$$(\mathrm{L}'') \qquad p=\min \xi; \qquad \sum_{j\in J} \beta_j g_1^j - \xi \leqq 0 \ (l=1, \ldots, m), \qquad (\beta, e)=1, \beta \geqq 0 .$$

Number p will be called the distance between C and R_-^m and it is clear that $C \cap R_-^m = \emptyset$ if and only if $p>0$. New vectors are successively added to the set defining C and the number α is enlarged in a similar way in both quasi-subgradient algorithms. Thus a sequence of convex hulls and a sequence of lower bounds for $\hat{h}$ are calculated analogously to Algorithm A.

Algorithm B

STEP 0: Set $\gamma>0$, $w^1 \in K''$, $D_0 \leftarrow \emptyset$, $\alpha_1 \leftarrow -\infty$. Choose a positive sequence $\{t_i\}_1^\infty$ such that $t_i \to 0$ and $\sum_{i=1}^{\infty} t_i = \infty$.

STEP 1: Determine any $x^i \in Q(w^i)$. Let $g^i = g(x^i)$.

STEP 2: If $f(x^i) > \alpha_i$, $\alpha_i \leftarrow f(x^i)$.

STEP 3: If $g^i \leqq 0$, then stop.

STEP 4: $D_i \leftarrow D_{i-1} \cup \{g^i\}$, $C_i \leftarrow \operatorname{co} D_i$.

STEP 5: If the distance between C_i and R_-^m, $p_i \leqq 0$, then stop.

STEP 6: Determine $\bar{d}^i \leftarrow g^i - (w^i, g^i)\, w^i$ and $d^i \leftarrow \bar{d}^i / \|\bar{d}^{-i}\|$.

STEP 7: $\tilde{w}^i \leftarrow w^i + \gamma t_i\, d^i$, $\tilde{w}^i \leftarrow \tilde{w}^i + q^i$, where $q_1^i = \begin{cases} -2\tilde{w}_1^i & \text{if } \tilde{w}_1^i < 0 \\ 0, & \text{otherwise.} \end{cases}$

STEP 8: $w^{i+1} \leftarrow \tilde{w}^i / \|\tilde{w}^i\|$.

STEP 9: $i \leftarrow i+1$. Go to Step 1.

Since Steps 1—3 are the same as in Algorithm A all comments relating to them are still valid. In Step 4 the new vector spanning the convex hull C_i is added and then in Step 5 the linear program (L'') is solved. If this stop test is satisfied then $\operatorname{co} R(\alpha_i) \cap R_-^m \neq \emptyset$ because $C_i \subset \operatorname{co} R(\alpha_i)$. Hence $\alpha_i = \hat{h}$. In Step 6 the normalized projection of g^i, orthogonal to w^i, is calculated. On Figure 2 the geometrical interpretation of Steps 6—8 is sketched.

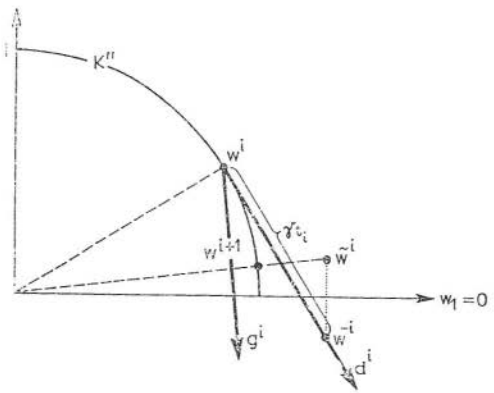

Fig. 2

The basic result relating to the convergence of Algorithm B was proved in [3].

PROPOSITION 8. If $w \in K''$, then $\lim_{i\to\infty} \inf (w, g^i) \leqq 0$, where g^i are generated in Algorithm B.

The above property implies

PROPOSITION 9. Algorithm B generates a sequence $\{p_i\}$ such that either i_0 exists: $p_{i_0} \leqq 0$, or $p_i \to 0$.

Proof. If for any i_0: $p_{i_0} \leqq 0$ then co $R(\alpha_{i_0}) \cap R_-^m \neq \emptyset$. This means $\alpha_{i_0} = \hat{h}$ and a solution of (SD'') has been found.

Suppose now that $p_i > 0$ for all i. Since the sequence $\{p_i\}$ is nonincreasing and bounded from below by zero it has as its limit $\bar{p} \geqq 0$. Suppose $\bar{p} \neq 0$. The obvious property: $C_j \subset C_i$ for all $j \leqq i$, implies that C_i and R_-^m can be strictly separated by a hyperplane. Hence, for some ρ'v exists such that $(v, w) \geqq \rho > 0$ for all $w \in C_i$ and $(v, w) \leqq 0$ for all $w \in R_-^m$. This means $w \in K''$ exists such that $(w, g_i) \geqq \rho$ for all i, which contradicts Proposition 8. Thus $\bar{p} = 0$. ∎

This result validates the main stop test in Step 5 but also implies the main convergence result.

PROPOSITION 10. Even if all $p_i > 0$, $\lim_{i\to\infty} \alpha_i = \hat{h}$.

Proof. Since $\{\alpha_i\}$ is nondecreasing and bounded from above it possesses a limit $\bar{\alpha}$. Suppose that $\bar{\alpha} < \hat{h}$. This means that co $R(\bar{\alpha}) \cap R_-^m = \emptyset$. On the other hand $\alpha_i \leqq \bar{\alpha}$ for all i implies $C_i \subset$ co $R(\bar{\alpha})$. Thus $\tilde{\rho} > 0$ exists such that $p_i \geqq \tilde{\rho}$ for all i. This means $\lim_{i\to\infty} p_i \geqq \tilde{\rho} > 0$, which contradicts Proposition 9. Thus $\bar{\alpha} = h$. ∎

The algorithm described below is also designed to obtain a solution of (SD''). The same linear program (L'') is solved as a stop test. Thus all notations remain the same.

Algorithm C

STEP 0 — 5: as in Algorithm B.

STEP 6: Determine $\bar{g}^i \leftarrow g^i + q^i$, where $q_l^i = \begin{cases} -g_l^i \text{ if } g_l^i < 0 \text{ and } w_l^i = 0. \\ 0 \text{ otherwise} \end{cases}$

STEP 7: Determine $\bar{d}^i \leftarrow \bar{g}^i - (w^i, \bar{g}^i)\, w^i$ and $d^i \leftarrow \bar{d}^i / \|\bar{d}^i\|$.

STEP 8: $\tau_i \leftarrow \min \{-w_l^i / d_l^i : 1 \in L_i\}$ where $L_i = \{1 : d_l^i < 0\}$
(if $L_i = \emptyset$ then $\tau_i \leftarrow +\infty$).

STEP 9: $\bar{w}^i \leftarrow w^i + \min(\gamma t_i, \tau_i)\, d^i$, $w^{i+1} \leftarrow \bar{w}^i / \|\bar{w}^i\|$.

STEP 10: $i \leftarrow i+1$. Go to Step 1.

When $\tau_i \geqq \gamma t_i$ and $\bar{g}^i = g^i$ both quasi-subgradient algorithms yield the same point as a new approximation. The difference appears when a face of the positive orthant becomes "acitve" i.e. for some i $w_l^i + \gamma t_i d_l^i < 0$. Uhile Algorithm B jumps back from such constraint: $w_l^{i+1} > 0$, Algorithm C puts the new approximation exactly on this hyperplane: $w_l^{i+1} = 0$. The latter allows also searching along such

"active" hyperplane, due to Step 6. The comparison of Figure 2 and Figure 3 points out the difference.

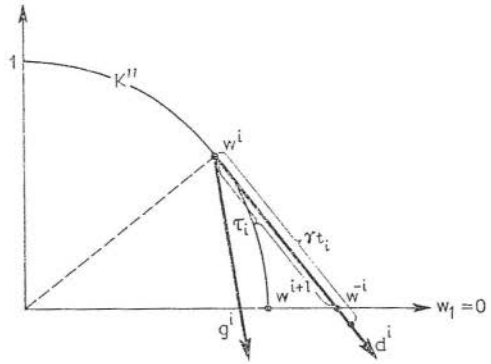

Fig. 3

Thus in Algorithm C another step siz sequence $\{\min(\gamma t_i, \tau_i)\}$ is use instead of $\{\gamma t_i\}$. Since R_+^m has obviously only a finite number of faces this modification can ravely affect the property that the step size series diverges. Hence the main convergence result for Algorithm B is usually valid for Algorithm C. Moreover the vector d^i generated in each iteration of Algorithm C has the same property of supporting level of h as g^i itself.

PROPOSITION 11. d^i determined in Step 7 is a quasi-subgradient of h at w^i.

Proof. Take any $u \in \{u \in R_+^m : h(u) > h(w^i)\} \neq \emptyset$. Denote $\mu = 1/\|\bar{d}^i\|$, $\mu > 0$. $(u, d^i) = \mu[(u, \bar{g}^i) - (w^i, \bar{g}^i)(u, w^i)] \geqq \mu[u, g^i) - (w^i, g^i)(u, w^i) \geqq \mu(u, g^i)]$. On the other hand $h(u) > h(w^i)$ means $h(u) > f(x^i)$, where $x^i \in Q(w^i)$. This implies $x^i \in X(u)$ i.e. $(u, g^i) > 0$, so $(u, d^i) > 0$ is a consequence. But Step 7 yields $(w^i, d^i) = 0$. As a result the desired inequality $(u, d^i) > (w^i, d^i)$ is obtained. ∎

5. Sub-problem solution

The solution of program (S) is required in each iteration of all described algof rithms. This minimization problem with one surrogate constraint in the case o-linear primal constraints and separable objective function takes the form of a knap-sack problem. The program is represented in "max" notation in conformity with common notation in integer programming.

$$(S') \qquad \max \sum_{k=1}^{n} \Phi_k(x_k); \quad \sum_{k=1}^{n} a_k x_k \leqq \delta, \quad 0 \leqq x_k \leqq u_k \text{ and integer } (k = 1, \ldots, n)$$

According to the primal problem: $\Phi_k = -f_k$, $[a_k]_{k=1, \ldots n} = a = wA$ and $\delta = (w, b)$. Some additional assumptions are required for the algorithm presented below

$\Phi_k(0)=0$, $a_k>0$ and $\delta \geqq 0$ are integer. They are not restrictive since any problem (S′) can be transformed to an equivalent one satisfyir g them. The algorithm proposed to solve (S′) is a modification of the first dynamic programming scheme presented in [4], Chapter 6. A short introduction is necessary before its presentation. By definition,

$$\Omega_\lambda(z)=\max \sum_{k=1}^{\lambda} \Phi_k(x_k);\ \sum_{k=1}^{\lambda} a_k x_k \leqq z,\ 0 \leqq x_k \leqq u_k \text{ and integer,}$$
$$\times (k=1, \ldots, \lambda) \text{ for } \lambda=2, \ldots, n \text{ and } z=0, 1, \ldots, \delta.$$

Isolating x_λ gives the recursive equation

$$\text{(R)} \qquad \Omega_\lambda(z)= \max_{x_\lambda=0,1,\ldots,\ \min\left(\left[\frac{z}{a_\lambda}\right],\ u_\lambda\right)} \left(\Phi_\lambda(x_\lambda)+\Omega_{\lambda-1}(z-a_\lambda x_\lambda)\right)$$

Adding the initial condition $\Omega_0(z)=0$ for $z=0, 1, \ldots, \delta$ allows us to extend (R) for $\lambda=1, \ldots, n$. Define also tableau $\Psi_\lambda(z)$:

$$\Psi_\lambda(z)=\begin{cases} 0 \text{ if } \Omega_\lambda(z)=\Omega_{\lambda-1}(z), \\ \bar{x}_\lambda \text{ if } \Omega_\lambda(z)>\Omega_{\lambda-1}(z) \text{ where } \bar{x}_\lambda \text{ denotes a solution of the maximization in (R).} \end{cases}$$

Algorithm S

STEP 0: Set $\lambda \leftarrow 0$, $\Omega_0(z) \leftarrow 0$, $z=0, 1, \ldots, \delta$

STEP 1: $\lambda \leftarrow \lambda+1$. For all $z<a_\lambda$ set $\Omega_\lambda(z) \rightarrow \Omega_{\lambda-1}(z)$ and $\Psi_\lambda(z) \leftarrow 0$. Set $z \leftarrow a_\lambda$.

STEP 2: Determine $v=\max\left(\Phi_\lambda(x_\lambda)+\Omega_{\lambda-1}(z-a_\lambda x_\lambda)\right)$,

$$x_\lambda=0, 1, \ldots, \min\left(\left[\frac{z}{a_\lambda}\right], u_\lambda\right).$$

If $v>\Omega_{\lambda-1}(z)$, then $\Omega_\lambda(z) \leftarrow v$ and $\Psi_\lambda(z) \leftarrow \bar{x}_\lambda$. Otherwise $\Omega_\lambda(z) \leftarrow \Omega_{\lambda-1}(z)$ and $\Psi_\lambda(z) \leftarrow 0$.

STEP 3: If $z<\delta$, let $z \leftarrow z+1$ and return to Step 2. If $z=\delta$ go to Step 4.

STEP 4: If $\lambda<n$, return to Step 1. If $\lambda=n$ go to Step 5.

STEP 5: $\hat{x}_\lambda \leftarrow \Psi_\lambda(z)$ and set $z \leftarrow z-a_\lambda \hat{x}_\lambda$.

STEP 6: If $\lambda>1$, then set $\lambda \leftarrow \lambda-1$ and return to Step 5. If $\lambda=1$, stop.

When the algorithm terminates, $\Omega_n(\delta)$ is the optimal value in program (S′) and $\hat{x}$ is an optimal solution. This means $\Omega_n(\delta)=-h(w)$. If (S′) has multiple optimal solutions, only one of them will be obtained by Algorithm S. However this is enough to procced with the computations in all the algorithms for calculating best surrogate constraints presented in this paper.

6. Numerical experiments

Several test problems were solved using Algorithms A, B and C. All of the test problems have the same form:

$$\min \sum_{k=1}^{n} f_k(x_k);\ \sum_{k=1}^{n} a_{lk} x_k \leqq b_l,\ l=1, \ldots, m,\ 0 \leqq x_k \leqq u_k \text{ and integer } (k=1, \ldots, n)$$

Each objective function is represented by a table $F=[f_{kj}]: f_{kj}=f_k(j)$ $j=1$ ..., u_k ($f_k(0)=0$). Elements of these tables are numbers randomly chosen from the interval $[-9.9; 0.0]$. Integer elements of matrix A were randomly chosen from the set $\{0, 1, ..., 9\}$. Vector b was determined in a such way that the unrestricted minima of the objective function are in feasible. All numerical data are included in an Appendix. Each test problem was solved using each algorithm for five different values of the parameters θ and γ: $\theta=0.2$, 0.4, 0.6, 0.8, 1.0 and $\gamma=0.25$, 0.5, 1.0, 2.0, 4.0. For finiteness of the algorithms in practice, the convergence criteria $g^i \leqq 0$, $r_i \leqq 0$. $p_i \leqq 0$ were replaced by $g^i < \varepsilon_1$, r_i, $p_i < \varepsilon_2$, respectively. The values $\varepsilon_1=10^{-4}$, $\varepsilon_2=10^{-5}$ were used for all runs during the experiments. Calculations were continued until the 75th iteration if an algorithm did not terminate with its stop test. In what follows the results of four problems are presented.

Test 1 ($n=5$, $m=3$) This problem is small and simple, but is a good example to show differences among the algorithms. Figure 4 shows a part of set K' (or rather

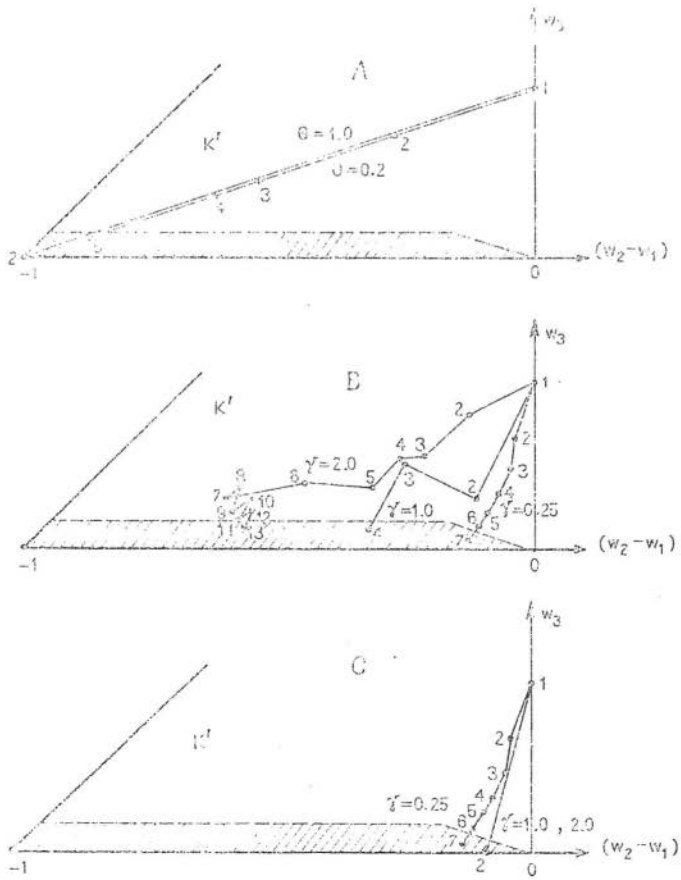

Fig. 4, 5, 6

its certain projection) with points w^i generated by Algorithm A for $\theta=0.2$ and 1.0, Figure 5 — points w^i normalized with L_1 norm, which have been generated by Algorithm B for $\gamma=0.25$, 1.0 and 2.0, Figure 6 — the same for Algorithm C ($\gamma=0.25$, 1.0 and 2.0). The interior of the dashed set is the optimal set for (SD'). The small number beside each point is an iteration number i.

Table 1 gives more details about the best runs.

Table 1

Alg.	1	2	3	4	5	6
A	$\theta=1.0$	−33.0	2	0.0	1	0.22
B	$\gamma=1.0$	−33.0	4	0.0	2	0.48
C	$\gamma=1.0$	−33.0	2	0.0	1	0.24

1 — value of θ or γ in the best run (sole criterion to compare runs is their number of iterations)
2 — maximal value of α_i achieved in the run
3 — number of iterations
4 — minimal value of stop test parameter r_i or p_i (the case r_i, $p_i<\varepsilon_2$ is expressed by 0.0)
5 — total number of simplex iterations in all stop test problems
6 — CPU time in seconds

Table 2 contains vectors w^i corresponding to the greatest values of $\alpha_i=h(w^i)$. For Algorithm 2 and 3 also L_1 normalisations of w^i are determined. Table 3 gives the solution of sub-problem (S) for these w^i.

Table 2

Alg.	w_i		
	1	2	3
A	1.0	0.0	0.0
B	0.8922	0.4486	0.0523
*	0.6404	0.3221	0.0375
C	0.7719	0.6357	0.0
*	0.5484	0.4516	0.0

* — $w/\|w\|_{L_1}$

Table 3

Alg.	x_k				
	1	2	3	4	5
A, B, C	0	3	1	3	1

, feasible in (P).

Thus program (SD) was solved with zero duality gap by all algorithms.

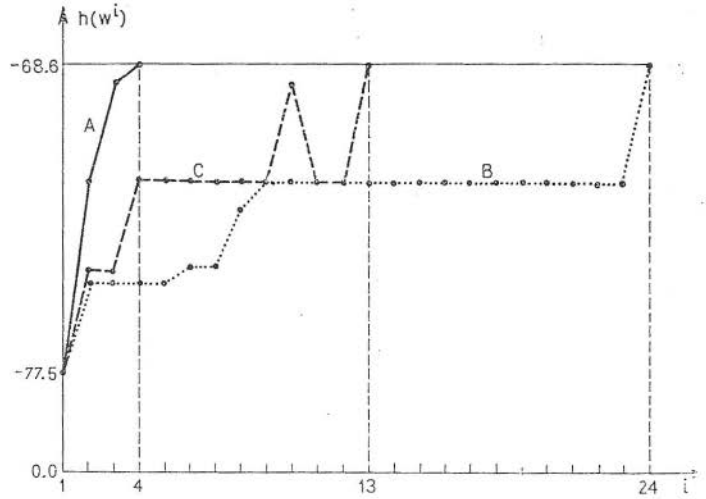

Fig. 7

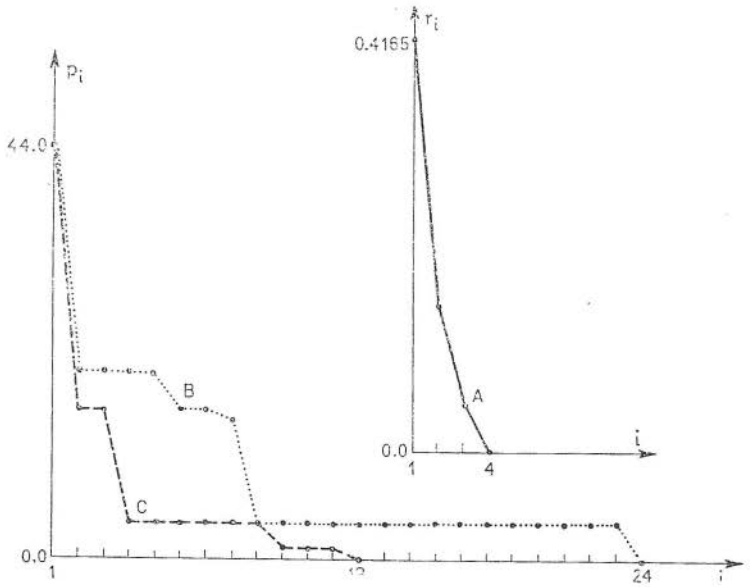

Fig. 8

Test 2 ($n=10$, $m=7$) This problem has moderate size, comparing with the tests in [11]. Figure 7 shows the values of $h(w^i)$ generated in the best runs of the algorithms. Figure 8 presents the sequences $\{r_i\}$, $\{p_i\}$ from these runs.
Table 4 compares results in an analogous way to Tab. 1. Tables 5 and 6 contain vectors w^i and x^i corresponding to the maximal value of $h(w^i)$.

Table 4

Alg.	1	2	3	4	5	6	
A	$\theta=0.6$	−68.6	4	0.0	1	2.07	———
B	$\gamma=4.0$	−68.6	24	0.0	8	12.15	.....
C	$\gamma=4.0$	−68.6	13	0.0	7	6.70	- - - - -

Table 5

Alg.	w_0						
	1	2	3	4	5	6	7
A	0.0041	0.0040	0.8698	0.0040	0.0040	0.110	0.0040
B	0.0872	0.0062	0.9945	0.0148	0.0002	0.0486	0.0272
*	0.0740	0.0053	0.8437	0.0126	0.0001	0.0412	0.0231
C	0.0	0.0	0.9972	0.0	0.0	0.0708	0.0237
*	0.0	0.0	0.9134	0.0	0.0	0.0649	0.0217

Table 6

Alg.	x_k									
	1	2	3	4	5	6	7	8	9	10
A, B, C	0.	3	1	1	1	2	1	1	1	3

, feasible in (P).

Program (SD) was solved with zero duality gap by all algorithms.

Test 3 ($n=15$, $m=10$) Figure 9 presents the sequences $\{h(w^i)\}$ from the best runs, while Figure 10 — the sequences $\{r_i\}$ and $\{p_i\}$.

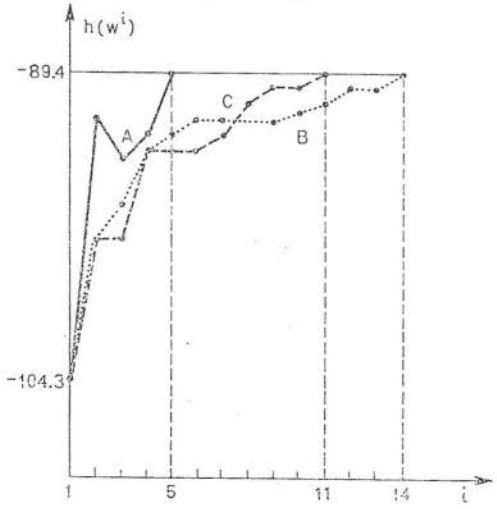

Fig. 9

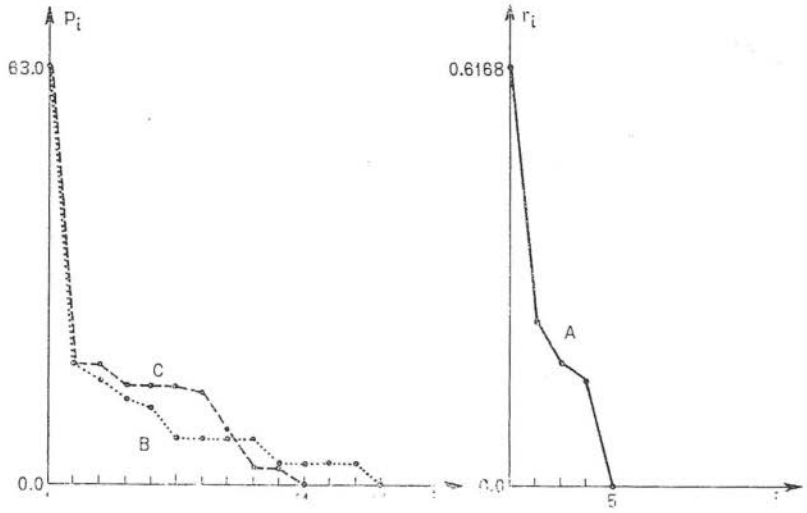

Fig. 10

Tables 7, 8 and 9 are analogous to Tables 1, 2 and 3, respectively.

Table 7

Alg.	1	2	3	4	5	6	
A	$\theta=0.6$	−89.4	5	0.0	2	5.48	———
B	$\gamma=1.0$	−89.4	14	0.0	10	15.31	·····
C	$\gamma=1.0$	−89.4	11	0.0	11	12.02	- - - - -

Table 8

Alg.	w_i									
	1	2	3	4	5	6	7	8	9	10
A	0.4334	0.0025	0.0025	0.0025	0.0025	0.0025	0.0025	0.0025	0.0250	0.5466
B	0.4993	0.0006	0.0154	0.0157	0.0091	0.0182	0.0082	0.0045	0.0227	0.8656
*	0.3422	0.0004	0.0105	0.0108	0.0062	0.0125	0.0056	0.0031	0.0155	0.5932
C	0.4559	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.1093	0.8833
*	0.3147	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0755	0.6098

Table 9

Alg.	x_k														
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
A, B, C	0	0	1	1	1	0	1	1	1	1	1	1	1	1	1

, feasible in (P).

Program (SD) was solved with zero duality gap by all algorithms.

Test 4 ($n=20$, $m=15$) The problem is the biggest one in the experiment but is still relatively small compared with real-life problems. The presentation of results is the same as in Test 3: Figures 11, 12 and Tables 10, 11, 12 give details of the best runs.

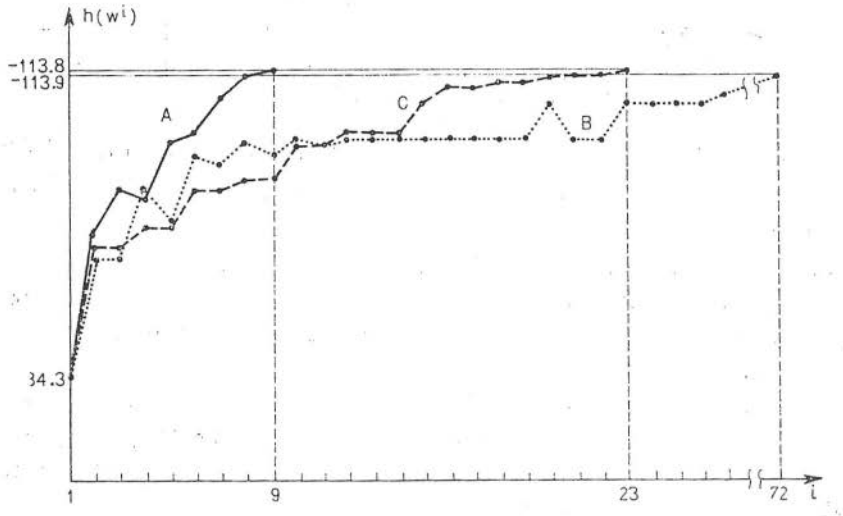

Fig. 11

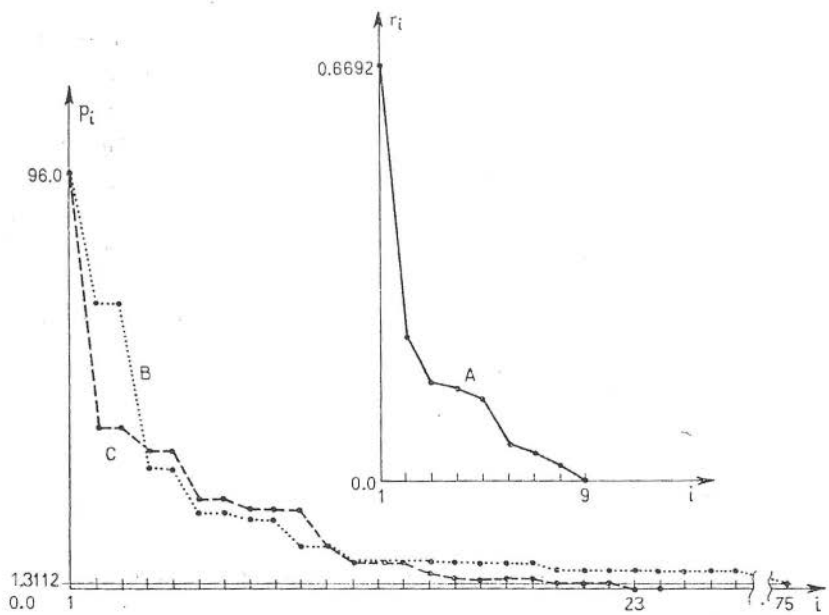

Fig. 12

Table 10

Alg.	1	2	3	4	5	6	
A	$\theta=0.4$	—113.8	9	0.0	12	18.39	———
B	$\gamma=2.0$	−113.9	75	1.3112	29	187.38	.....
C	$\gamma=2.0$	−113.8	23	0.0	28	47.28	-------

Progran (SD) was solved with a non-zero duality gap by Algorithm A and Algorithm C. The sequence $\{\alpha_t\}$ did not reach $\hat{h}$ in Algorithm B before the 75th iteration.

A few words should perhaps be said about starting points and linear programming for solving (L′) and (L″). In all runs of Algorithm A $w_l^1=1/m$, $l=1, ..., m$ was used. $w_l^1=1/m$, $l=1, ..., m$ was set for Algorithm B and C. These starting points are neutral and they do not prefer any primal constraints. If some information about "activity" of constraints is available before initiation of the algorithm then some of them can be neglected by setting $w_l^1=0$ for all coordinates corresponding to the "non-binding" ones. This can be especially valuable for Algorithm C because it leads to a searching in a subset of the dual feasible set having smaller dimension. If a starting point is chosen closer to the dual optimal set then probably smaller values for γ will work better. In general Algorithm A is less sensitive to the choice of starting point than Algorithms B and C.

Programs (L′) and (L″) have been solved in all algorithms by the simplex procedure taking advantage of the fact that in the subsequent iteration only one constraint or one variable is added. Thus the optimal simplex tableau from the preceding iteration can be used to construct an initial tableau for the next one. Then sometimes it is not necessary to do any simplex iterations because the proceeding solution is still optimal. Moreover in the first iteration of all presented algorithms only one vector g^1 generates the linear program so the optimal simplex tableau can be defined directly without any computations.

The numerical experiments were done at the Computer Center of University of Hiroshima using HITAC M 200-H.

7. Conclusions

Taking all our results into account the presented algorithms proved their usefulness as a tool for solving the surrogate dual program, which may be considered a difficult one, since it involves maximizing a quasiconcave, often discontinuous,

Table 11

Alg.	w_i														
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
A	0.0009	0.0009	0.0009	0.7038	0.0009	0.0395	0.2461	0.0090	0.0009	0.0009	0.0009	0.0009	0.0009	0.0009	0.0009
B	0.0013	0.0061	0.0024	0.9361	0.0009	0.0367	0.3494	0.0051	0.0004	0.0012	0.0105	0.0020	0.0064	0.0020	0.0024
*	0.0009	0.0045	0.0018	0.6868	0.0006	0.0269	0.2564	0.0037	0.0003	0.0009	0.0077	0.0015	0.0047	0.0015	0.0018
C	0.0195	0.0	0.0	0.9332	0.0	0.0939	0.3462	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
*	0.0140	0.0	0.0	0.6699	0.0	0.0674	0.2487	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

Table 12

Alg.	x_k																			
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
A, C	0	3	1	1	1	2	0	1	0	1	1	1	1	1	1	2	1	1	0	0
B	0	3	1	1	1	2	1	1	1	1	1	1	1	1	1	0	1	0	0	0

[392]

, both vectors are in feasible in (SP).

function. The algorithms are based two on fundamentally different methods. The first originates from generalised programming while the another imitates subgradient methods. Algorithm A, representing the first approach, seems to be better organized but needs more calculations because the linear program is its immanent element. Algorithms B and C use a linear program as a stop test which is more effective then, but could be replaced by, if the problem is too large a simpler heuristic test. However both are more local in their nature. Since only local information about h is applied in the form of a quasi-subgradient, it is very difficult to avoid zigzagging and this undersirable feature appears in Algorithm B and C. Additionally the step size sequence is chosen arbitrarily and can not (theoretically) be modified during calculations. In fact, it is changed in Algorithm C but in very rough way. Thus the simplicity of these algorithms remains their most important advantage for large real life problems. Algorithm A collects the information in the form of a polytope which is used to derive the next approximation of the solution. Moreover the upper level sets of the surrogate dual function are also polytopes in the case of linear primal constraints so the form of approximation used is more adequate in this case. These two facts may have caused the superiority of Algorithm A in the numerical experiments. The relatively small size of all test problems may have also affected the comparison. The results of the experiments indicate the following hierarchy among the algorithms: Algorithm C is better than Algorithm B but both are dominated by Algorithm A. In the latter a moderate choice of parameter θ i.e. around 0.5, seems to be appropriate. Either choosing θ too small or too close to one causes a slower convergence process. The choice of θ near one even leads sometimes to behaviour similar to zigzagging. Nevertheless this algorithm is less sensitive to in appropriate choice of 0 than are Algorithm is less sesnitive to inappropriate choito of θ than are Algorithms B and C to choice of γ. Also, properties of the problem are more important for a proper selection of γ in quasi-subgradient schemes. If zigzagging dominates during searching, then bigger values for γ work better, but when the approximations approach the optimal set, diminishing of γ seems to be necessary. Unfortunately there is no way to do this in these algorithms. However $\gamma=1$ or 2 are not bad choices in many of the test problems. Thus the decaying polytope algorithm is recomended when the size of the problem allows us to store all the vectors g^i and proceed with solving the linear program at each iteration. If this is not possible, the quasi-subgradient algorithm remains as an alternative. The second version (Algorithm C) is more effective when the dual optimal set lies close to the boundary of K'', which means that a group of primal constraints can, in fact, be omitted. Since such a situation is common in real life problems, Algorithm C may be preferable.

References

[1] BRICKER D.L. Bounding a class of nonconvex linearly-constrained resource allocation problems via the surrogate dual. *Mathematical Programming*, **18** (1980), 68–83.

[2] COOPER M.W. The use of dynamic programming methodology for the solution of a class of nonlinear programming problems. *Naval Research Logistics Quarterly*, **27** (1980) 1, 89–96.

[3] DYER M.E. Calculating surrogate constraints. *Mathematical Programming*, **19** (1980), 255–278.

[4] GARFINKEL R.S., NEMHAUSER G.L. Integer Programming. New York, John Wiley&Sons 1972.

[5] GLOVER F. Surrogate constraints. *Operations Research*, **16** (1968), 741–749.

[6] GLOVER F. Surrogate constraints duality in mathematical programming. *Operations Research*, **23** (1975), 434–451.

[7] GRRENBERG H.J., PIERSKALLA W.P. Surrogate mathematical programs. *Operations Research*, **18** (1970), 924–939.

[8] KARWAN M.H., RARDIN R.L. Some relationships between Lagrangian and surrogate duality in integer linear programming. *Mathematical Programming*, **17** (1979), 320–334.

[9] KARWAN M.H., RARDIN R.L. Surrogate duality in a branch-and-bound procedure. *Naval Research Logistic Quaterly*, **28** (1981) 1, 93–101.

[10] LUENBERGER D.G. Quasiconvex programming. *SIAM Journal of Applied Mathematics*, **16** (1968), 1090–1095.

[11] MORIN T.L., MARSTEN R.E. An algorithm for nonlinear knapsack problems. *Management Sciences*, **22** (1976), 1147–1158.

Received, August 1983.

9. Appendix

Test 1

$u_k = 5,\ k = 1, \ldots, 5$

f_{kj}	j = 1	2	3	4	5
1	0.0	−6.3	−8.9	−5.7	−7.3
2	−1.1	−2.8	−7.5	−5.6	−1.2
k 3	−7.5	−2.7	−4.1	−6.4	−7.5
4	−8.0	−1.7	−8.9	−5.8	−1.1
5	−9.1	−7.3	−2.3	−8.5	−6.6

: F

a_{lk}	k = 1	2	3	4	5	b_l
1	9	5	7	2	6	41
l 2	7	2	4	0	5	29
3	4	8	5	1	1	38

: A : b

Test 2

$u_k = 5,\ k = 1, \ldots, 10$

f_{kJ}	j = 1	2	3	4	5
1—5	like in Test 1				
6	−0.1	−8.1	−2.7	−0.8	−4.8
7	−5.2	−0.1	−4.6	−4.3	−8.8
8	−9.4	−2.0	−3.4	−2.3	−7.5
9	−6.3	−5.9	−8.4	−5.8	−3.6
10	−4.5	−3.3	−7.5	−6.0	−7.5

k : F

a_{lk}	k = 1	2	3	4	5	6	7	8	9	10	b_l
1	1	1	2	3	7	3	9	3	1	0	61
2	9	6	4	6	4	8	1	0	6	4	82
3	6	2	3	3	7	6	2	9	5	3	58
4	0	1	5	4	4	4	6	5	0	8	71
5	5	0	9	0	2	1	0	8	5	0	51
6	0	4	2	8	9	8	6	1	0	7	75
7	9	4	3	0	6	3	4	1	4	7	68

l : A ; b

Test 3

$u_k = 5,\ k = 1, \ldots, 15$

f_{kJ}	j = 1	2	3	4	5
1—10	like in Test 2				
11	−8.7	−6.3	−9.1	−1.9	−7.4
12	−7.0	−5.5	−2.2	−1.9	−1.5
13	−8.2	−5.8	−2.9	−2.4	−4.4
14	−9.4	−1.9	−5.6	−9.3	−9.8
15	−6.1	−6.6	−2.3	−4.7	−7.7

k : F

a_{lk}	k = 1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	b_l
1	6	2	2	5	7	3	8	4	7	3	4	5	6	5	7	69
2	3	6	4	8	6	4	6	3	5	6	9	3	7	2	2	88
3	1	3	3	3	1	6	5	9	8	5	9	9	6	4	6	93
4	1	0	0	2	7	1	9	1	1	0	9	2	5	8	4	68
5	2	7	8	9	6	8	3	0	2	1	7	3	2	0	0	85
6	2	6	9	4	8	7	4	6	5	1	5	9	0	7	0	86
7	1	9	9	5	7	2	5	8	6	9	1	9	0	3	6	83
8	8	2	8	0	1	1	6	9	8	6	9	9	0	1	1	74
9	7	4	8	2	4	4	6	7	3	9	5	5	0	3	4	71
10	5	8	4	3	7	9	4	6	0	3	5	9	3	5	2	51

l : A ; b

Test 4

$$u_k = 5,\ k = 1, \ldots, 20$$

f_{kj}	j				
	1	2	3	4	5
1—15	like in test 3				
16	−0.9	−7.0	−0.8	−8.2	−7.5
17	−8.9	−4.1	−9.6	−0.7	−1.3
18	−4.4	−1.0	−0.2	−9.7	−3.0
19	−1.4	−7.6	−7.7	−6.7	−3.3
20	−0.3	−3.3	−7.3	−5.7	−7.5

k : F

a_{lk}	k																				b_l
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	
1	2	8	9	2	0	7	6	1	8	4	8	3	7	3	4	1	4	4	3	0	91
2	3	5	2	4	8	8	8	2	3	1	2	8	7	9	8	0	9	8	5	9	110
3	2	3	0	3	1	8	5	6	0	1	7	9	3	5	6	9	0	5	0	0	108
4	9	5	6	2	8	7	9	7	9	5	2	7	8	2	2	3	7	5	8	2	96
5	8	8	2	6	2	1	0	9	5	6	9	8	1	4	9	3	1	2	8	8	105
6	0	7	5	8	6	9	6	2	7	7	4	9	9	2	0	1	3	8	9	8	103
7	4	0	9	4	6	1	2	5	4	8	7	7	8	6	5	9	9	4	1	8	99
8	2	6	4	2	8	8	7	8	1	0	8	4	1	6	2	2	4	7	0	8	97
9	5	8	3	7	0	8	6	9	7	4	5	9	3	6	2	0	2	0	3	0	100
10	2	9	7	7	1	7	2	1	0	1	8	3	1	1	3	6	2	2	6	2	93
11	2	0	2	3	4	8	2	1	5	8	8	0	2	1	0	1	3	7	2	3	101
12	7	7	3	9	9	1	7	2	5	1	9	7	0	7	2	3	4	5	1	2	92
13	2	9	0	2	3	8	4	2	5	7	4	0	1	5	0	0	4	1	5	0	97
14	1	5	1	4	7	3	7	8	7	8	7	7	3	2	0	8	2	3	5	9	108
15	4	2	7	5	5	1	6	1	7	9	3	9	3	2	0	5	2	6	2	7	95

l : A : b

Trzy algorytmy wyznaczania ograniczeń zastępczych w zadaniach programowania calkowitoliczbowego

Dualne zadanie wyznaczania ograniczenia zastępczego pozwala zazwyczaj uzyskać odstęp dualności mniejszy niż dla zadania dualnego Lagrange'a .Oferuje ono zatem możliwość otrzymania lepszego oszacowania wartości optymalnej w metodach podziału i oszacowań.

Przedstawiono trzy algorytmy rozwiązywania zadania wyznaczania ograniczenia zastępczego. Jeden oparty jest na schemacie wykorzystującym ciąg aproksymacji wielościennych, który został zaproponowany w [3]. Dwa pozostałe są wariantami metody poszukiwań w kierunku z ustalonym ciągiem współczynników kroku. W obu wykorzystano sprawny test stopu oparty na pewnej charakterystyce dualnej wartości optymalnej.

Zadanie załadunku z jednym ograniczeniem rozwiązywane jest w każdej iteracji w wszystkich trzech algorytmach. Przedstawiono wyniki testu obliczeniowego.

Три алгоритма для вычисления заместительных ограничении в задачах целочисленного программирования

Двойственная задача вычисления заместителного ограничения предлогает более эффективную оценку для оптималного значения в методах ветвей и границ, потому что расстояние двойственности для этой задачи обычно меньше чем для задачи Лагранжа.

Три алгоритма представлены для разрешения проблемы. Вычислительный эксперимент служит оравнению их свойств. Дза алгоритма пользуюсь квази-субградиентом в процессе строения направления поиска. Один из них был описап в [3] но без полезной критерьи остоновления которая здесь представлена. Третий возникает от идейи обобщенного программирования и он основан на схеме описаной в [3].

Во всех представленых здесь алгоритмах в каждой итерации надо решать задачу „рюкзака" с одном ограничением.