M.Gyulassy and H.Harlander (1991) Elastic tracking and neural network algorithms for complex pattern recognition. *Computer Physics Comm.*, **66**, 31.

M.Ohlsson, C.Peterson and B.Söderberg (1991) Neural networks for optimization problems with inequality constraints—the knapsack problem. *Neural Computation*, (submitted).

W.P.Press, B.P.Flannery, S.A.Teukolsky and W.T.Vettering (1986) *Numerical Recipes, The Art of Scientific Computing.* Cambridge University Press, Cambridge.

Chapter 6

Lagrangean Relaxation

John E Beasley

6.1 Introduction

As was remarked in chapter 1 of this book, finding good solutions to hard problems in combinatorial optimization requires the consideration of two issues:

- calculation of an upper bound that is as close as possible to the optimum;

- calculation of a lower bound that is as close as possible to the optimum.

General techniques for generating good upper bounds are essentially heuristic methods of the type considered in the preceding chapters of this book. In addition, for any particular problem, we may well have techniques which are specific to the problem being solved.

On the question of lower bounds, one well-known general technique which is available is Linear Programming (LP) relaxation. In LP relaxation we take an integer (or mixed-integer) programming formulation of the problem and relax the integrality requirement on the variables. This gives a linear program which can either be:

- solved exactly using a standard algorithm (simplex or interior point); or

- solved heuristically (dual ascent).

The solution value obtained for this linear program gives a lower bound on the optimal solution to the original problem. We shall illustrate both of these approaches in this chapter.

Another well-known (and well-used) technique which is available to find lower bounds is Lagrangean relaxation. This technique will be expounded at much greater length. Suffice to say for the moment that Lagrangean relaxation involves:

1. taking an integer (or mixed-integer) programming formulation of the problem;

2. attaching Lagrange multipliers to some of the constraints in this formulation and relaxing these constraints into the objective function;

3. solving (exactly) the resulting integer (or mixed-integer) program.

The solution value obtained from step 3 above gives a lower bound on the optimal solution to the original problem. At first sight this might not appear to be a useful approach since at step 1 above we have an integer (or mixed-integer) programming formulation of the problem and we propose to generate a lower bound for it by solving another integer (or mixed-integer) program (step 3).

There are two basic reasons why this approach is well-known (and well-used).

- Many combinatorial optimization problems consist of an easy problem (in the NP-complete sense, i.e. solvable by a polynomially bounded algorithm) complicated by the addition of extra constraints. By absorbing these complicating constraints into the objective function (step 2) we are left with an easy problem to solve and attention can then be turned to choosing numerical values for the Lagrange multipliers.

- Practical experience with Lagrangean relaxation has indicated that it gives very good lower bounds at reasonable computational cost.

Choosing values for the Lagrange multipliers is of key importance in terms of the quality of the lower bound generated (we much prefer

lower bounds which are close to the optimal solution). Two general techniques are available here—*subgradient optimization* and *multiplier adjustment*.

Readers are reminded here of the description and diagrams (Figures 1.1, 1.2) given in chapter 1. For finding upper bounds we have heuristic techniques such as those described in chapters 2-5. For finding lower bounds we have the various relaxations mentioned above.

6.2 Overview

6.2.1 Techniques

In this chapter we deal with:

- Lagrangean relaxation
- Lagrangean heuristics
- problem reduction
- subgradient optimization
- multiplier adjustment
- dual ascent
- tree search

These techniques are building blocks for constructing optimal solution (exact) algorithms, that is algorithms which guarantee to find the optimal solution to a particular problem.

But guaranteeing to find the optimal solution is not sufficient. An algorithm that guarantees to find the optimal solution, but which takes days to run on a computer, is obviously of much less use than an algorithm which guarantees to find the optimal solution but which takes only minutes to run on a computer.

This issue of computational effectiveness, i.e. building an algorithm that can solve problems in a reasonable time frame, is a key one in algorithm design and development. Therefore, in this chapter, we also have as an implicit objective the design of a computationally effective optimal solution algorithm incorporating the above techniques. In order to do this we shall consider one of the simplest

NP-complete combinatorial optimization problems—the set covering problem.

6.2.2 Review of the literature

Much of what is contained in this chapter is historical—that is, it has been known for a number of years. To enable the reader to get some insight into current up-to-date work involving Lagrangean relaxation we have adopted the strategy of:

- selecting calendar year 1991 (the most recent complete calendar year at the time of writing);

- selecting a number of journals; and

- reviewing all papers in those journals in our selected year which use Lagrangean relaxation.

This will provide a systematic insight into the range of problems to which Lagrangean relaxation is being applied.

6.3 Basic Methodology

6.3.1 Introduction

Consider the following general zero-one problem (written in matrix notation) which we shall refer to as problem (P):

$$
\begin{array}{ll}
\text{minimize} & cx \\
\text{subject to} & Ax \geq b \\
& Bx \geq d \\
& x \in (0,1)
\end{array}
$$

Note here that although we deal in this chapter purely with zero-one integer programs the material presented is equally applicable both to pure (general) integer programs and to mixed-integer programs.

As mentioned above, one way to generate a lower bound on the optimal solution to problem (P) is via the linear programming relaxation. This entails replacing the integrality constraint $[x \in (0,1)]$ by its linear relaxation $[0 \leq x \leq 1]$ to give the following linear program:

$$
\begin{array}{ll}
\text{minimize} & cx \\
\text{subject to} & Ax \geq b \\
& Bx \geq d \\
& 0 \leq x \leq 1
\end{array}
$$

This linear program can be solved exactly using a standard algorithm (e.g. simplex or interior point) and the solution value obtained gives a lower bound on the optimal solution to the original problem (P).

In many cases however solving the linear programming relaxation of (P) is impracticable, because typically (P) involves a large (often extremely large) number of variables and/or constraints. We therefore need alternative techniques for generating lower bounds.

6.3.2 Lagrangean relaxation

Lagrangean relaxation was developed in the early 1970s with the pioneering work of Held and Karp [1, 2] on the travelling salesman problem and is today an indispensable technique for generating lower bounds for use in algorithms to solve combinatorial optimization problems.

We define the Lagrangean relaxation of problem (P) with respect to the constraint set $Ax \geq b$ by introducing a Lagrange multiplier vector $\lambda \geq 0$ which is attached to this constraint set and brought into the objective function to give:

$$
\begin{array}{ll}
\text{minimize} & cx + \lambda(b - Ax) \\
\text{subject to} & Bx \geq d \\
& x \in (0,1)
\end{array}
$$

What we have done here is:

- to choose some set of constraints in the problem for relaxation; and

- to attach Lagrange multipliers to these constraints in order to bring them into the objective function.

The key point is that the program we are left with after Lagrangean relaxation, for any $\lambda \geq 0$, gives a lower bound on the optimal solution to the original problem (P). This can be seen as follows.

The value of

$$\begin{aligned} \text{minimize} \quad & cx \\ \text{subject to} \quad & Ax \geq b \\ & Bx \geq d \\ & x \in (0,1) \end{aligned}$$

is greater than the value of

$$\begin{aligned} \text{minimize} \quad & cx + \lambda(b - Ax) \\ \text{subject to} \quad & Ax \geq b \\ & Bx \geq d \\ & x \in (0,1) \end{aligned}$$

since as $\lambda \geq 0$ and $(b - Ax) \leq 0$ we are merely adding a term which is ≤ 0 to the objective function. This in turn is greater than the value of

$$\begin{aligned} \text{minimize} \quad & cx + \lambda(b - Ax) \\ \text{subject to} \quad & Bx \geq d \\ & x \in (0,1) \end{aligned}$$

since removing a set of constraints from a minimization problem can only reduce the objective function value.

The program after Lagrangean relaxation, namely:

$$\begin{aligned} \text{minimize} \quad & cx + \lambda(b - Ax) = (c - \lambda A)x + \lambda b \\ \text{subject to} \quad & Bx \geq d \\ & x \in (0,1) \end{aligned}$$

can be called the Lagrangean lower bound program (LLBP) since, as shown above, it provides a lower bound on the optimal solution to the original problem (P) for any $\lambda \geq 0$. Note that the above proof that Lagrangean relaxation generates lower bounds is quite general, i.e. the constraints/objective function need not be linear functions. ◀

There are two key issues highlighted by the above Lagrangean relaxation:

- a *strategic* issue—namely, why did we choose to relax the set of constraints $Ax \geq b$ when we could equally well have chosen to relax $Bx \geq d$?

- a *tactical* issue—namely, how can we find numerical values for the multipliers?

In particular note here that we are interested in finding the values for the multipliers that give the maximum lower bound, i.e. the lower bound that is as close as possible to the value of the optimal integer solution. This involves finding multipliers which correspond to:

$$\max_{\lambda \geq 0} \left\{ \begin{aligned} \text{minimize} \quad & cx + \lambda(b - Ax) \\ \text{subject to} \quad & Bx \geq d \\ & x \in (0,1) \end{aligned} \right\}$$

This program is called the *Lagrangean dual program*.

Ideally the optimal value of the Lagrangean dual program (a maximization program) is equal to the optimal value of the original zero-one integer program (a minimization problem). If the two programs do not have optimal values which are equal then a *duality gap* is said to exist, the size of which is measured by the (relative) difference between the two optimal values.

In order to illustrate Lagrangean relaxation we shall now consider one of the simplest NP-complete combinatorial optimization problems—the set covering problem.

6.3.3 Set covering problem

The set covering problem (SCP) is the problem of covering the rows of an m-row, n-column, zero-one matrix $[a_{ij}]$ by a subset of the columns at minimum cost.

Defining $x_j = 1$ if column j (cost $c_j > 0$) is in the solution
$ = 0$ otherwise

the SCP is:

$$\begin{aligned} \text{minimize} \quad & \sum_{j=1}^{n} c_j x_j \\ \text{subject to} \quad & \sum_{j=1}^{n} a_{ij} x_j \geq 1; \quad i = 1, \ldots, m \\ & x_j \in (0,1); \quad j = 1, \ldots, n \end{aligned}$$

The first set of constraints in this program ensures that each row is covered by at least one column and the second set form the integrality constraints.

In order to generate a Lagrangean relaxation of this SCP we need:

1. to choose some set of constraints in the problem for relaxation; and

2. to attach Lagrange multipliers to these constraints in order to bring them into the objective function.

Step 1 above is not usually an easy step. As commented above, the choice of which set of constraints to relax is a strategic issue. However, for the SCP we simply have one distinct set of constraints $(\sum_{j=1}^{n} a_{ij}x_j \geq 1; i = 1,\ldots,m)$ and so we choose this set of constraints for relaxation and attach Lagrange multipliers $\lambda_i \geq 0$ $(i = 1,\ldots,m)$ to these constraints.

If we do this we find that LLBP is:

$$\text{minimize} \quad \sum_{j=1}^{n} c_j x_j + \sum_{i=1}^{m} \lambda_i (1 - \sum_{j=1}^{n} a_{ij}x_j)$$
$$\text{subject to} \quad x_j \in (0,1); \quad j = 1,\ldots,n$$

i.e.

$$\text{minimize} \quad \sum_{j=1}^{n} [c_j - \sum_{i=1}^{m} \lambda_i a_{ij}]x_j + \sum_{i=1}^{m} \lambda_i$$
$$\text{subject to} \quad x_j \in (0,1); \quad j = 1,\ldots,n$$

Defining $C_j = [c_j - \sum_{i=1}^{m} \lambda_i a_{ij}]$, $j = 1,\ldots,n$, we find that LLBP becomes:

$$\text{minimize} \quad \sum_{j=1}^{n} C_j x_j + \sum_{i=1}^{m} \lambda_i$$
$$\text{subject to} \quad x_j \in (0,1); \quad j = 1,\ldots,n$$

Now the solution (X_j) to LLBP can be found by inspection, namely:

$$X_j = 1 \quad \text{if } C_j \leq 0$$
$$= 0 \quad \text{otherwise}$$

with the solution value (Z_{LB}) of LLBP being given by:

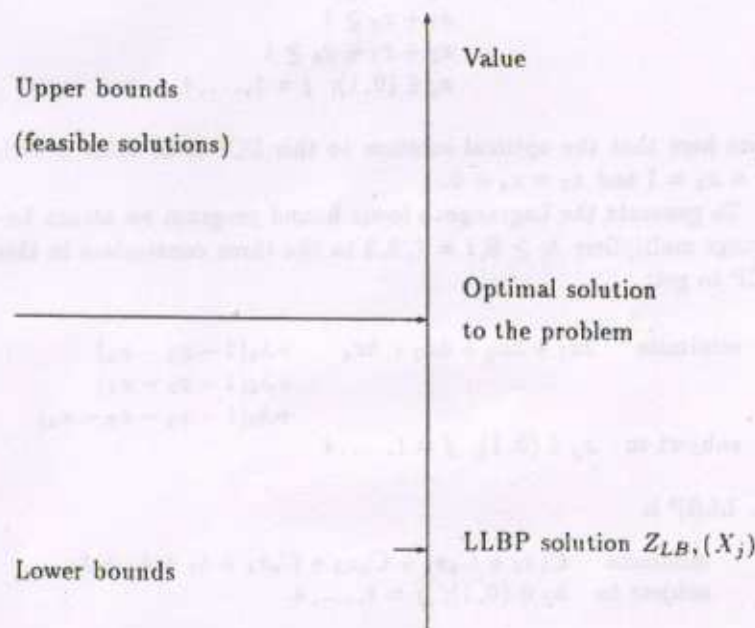$$Z_{LB} = \sum_{j=1}^{n} C_j X_j + \sum_{i=1}^{m} \lambda_i$$

Figure 6.1: LLBP solution

where Z_{LB} is a lower bound on the optimal solution to the original SCP.

Figure 6.1 summarizes the situation. In that figure we have a point on the value line (a lower bound) associated with the solution $(Z_{LB}, (X_j))$ to LLBP.

To illustrate the Lagrangean relaxation of the SCP given above consider the following example SCP (with 3 rows and 4 columns) defined by:

$$(c_j) = (2,3,4,5)$$

$$[a_{ij}] = \begin{bmatrix} 1010 \\ 1001 \\ 0111 \end{bmatrix}$$

i.e. the example SCP is:

$$\begin{aligned}
\text{minimize} \quad & 2x_1 + 3x_2 + 4x_3 + 5x_4 \\
\text{subject to} \quad & x_1 + x_3 \geq 1 \\
& x_1 + x_4 \geq 1 \\
& x_2 + x_3 + x_4 \geq 1 \\
& x_j \in (0,1); \quad j = 1,\dots,4
\end{aligned}$$

Note here that the optimal solution to this SCP is of value 5 with $x_1 = x_2 = 1$ and $x_3 = x_4 = 0$.

To generate the Lagrangean lower bound program we attach Lagrange multipliers $\lambda_i \geq 0; i = 1,2,3$ to the three constraints in this SCP to get:

$$\begin{aligned}
\text{minimize} \quad & 2x_1 + 3x_2 + 4x_3 + 5x_4 && +\lambda_1(1 - x_1 - x_3) \\
& && +\lambda_2(1 - x_1 - x_4) \\
& && +\lambda_3(1 - x_2 - x_3 - x_4) \\
\text{subject to} \quad & x_j \in (0,1); \quad j = 1,\dots,4
\end{aligned}$$

i.e. LLBP is

$$\begin{aligned}
\text{minimize} \quad & C_1 x_1 + C_2 x_2 + C_3 x_3 + C_4 x_4 + \lambda_1 + \lambda_2 + \lambda_3 \\
\text{subject to} \quad & x_j \in (0,1); \quad j = 1,\dots,4
\end{aligned}$$

where

$$\begin{aligned}
C_1 &= (2 - \lambda_1 - \lambda_2) \\
C_2 &= (3 - \lambda_3) \\
C_3 &= (4 - \lambda_1 - \lambda_3) \\
C_4 &= (5 - \lambda_2 - \lambda_3)
\end{aligned}$$

Hence (X_j), the solution values of the (x_j), are given by

$$\begin{aligned}
X_j &= 1 \quad \text{if } C_j \leq 0 \\
&= 0 \quad \text{otherwise}
\end{aligned}$$

with the solution value for LLBP (Z_{LB})—a valid lower bound on the optimal solution to the original SCP—being given by

$$Z_{LB} = C_1 X_1 + C_2 X_2 + C_3 X_3 + C_4 X_4 + \lambda_1 + \lambda_2 + \lambda_3$$

6.3.4 Example Lagrange multiplier values

As commented above the choice of numerical values for the Lagrange multipliers is a tactical issue. For the moment consider the (arbitrarily decided) set of values for the Lagrange multipliers of:

$$\lambda_1 = 1.5$$

$$\lambda_2 = 1.6$$

$$\lambda_3 = 2.2$$

then

$$C_1 = (2 - \lambda_1 - \lambda_2) = -1.1$$

$$C_2 = (3 - \lambda_3) = 0.8$$

$$C_3 = (4 - \lambda_1 - \lambda_3) = 0.3$$

$$C_4 = (5 - \lambda_2 - \lambda_3) = 1.2$$

The solution to LLBP is

$$X_1 = 1, X_2 = X_3 = X_4 = 0$$

from which it is easily seen that $Z_{LB} = -1.1 + 0 + 0 + 0 + 1.5 + 1.6 + 2.2 = 4.2$, which is indeed a lower bound on the optimal solution (which we know is of value 5) to the original SCP.

6.3.5 Advanced Lagrangean relaxation

The basic Lagrangean relaxation methodology can be extended in several ways. In this section we draw some of these results together, and comment on their implications.

(1) If we relax equality constraints then λ_i is unrestricted in sign (i.e. λ_i can be positive or negative).

(2) A common fallacy in Lagrangean relaxation is to believe that, if the solution to LLBP is feasible for the original problem, then it is also optimal for the original problem. This is <u>incorrect</u>.

For example consider the SCP with 3 rows and 4 columns dealt with above. Set $\lambda_1 = \lambda_2 = \lambda_3 = 10$ and solve LLBP. The solution is $X_1 = X_2 = X_3 = X_4 = 1$. This is certainly a feasible solution for the original problem (the SCP) but by no means the optimal solution!

Under what circumstances therefore does the solution to LLBP being feasible for the original problem also imply that it is optimal for the original problem?

The answer to this question is simple. Consider LLBP:

$$\text{minimize} \quad cx + \lambda(b - Ax)$$
$$\text{subject to} \quad Bx \geq d$$
$$x \in (0, 1)$$

Suppose that the Lagrange multipliers $\lambda \geq 0$ are such that the solution X to LLBP is feasible for the original problem (i.e. X satisfies $AX \geq b$, $BX \geq d$ and $X \in (0,1)$). The value of this feasible solution is cX whereas the value of the lower bound obtained from LLBP is $[cX + \lambda(b - AX)]$. Then if these two values coincide, i.e. the upper bound cX is equal to the lower bound $[cX + \lambda(b - AX)]$, X is optimal.

In other words a solution X to a Lagrangean lower bound program is optimal for the original problem *only if*:

1. X is feasible for the original problem; and

2. $cX = [cX + \lambda(b - AX)]$, i.e. $\lambda(b - AX) = 0$

The reason why the fallacy referred to above has appeared is clear. If we are relaxing *equality* constraints $(Ax = b)$ then any solution to the Lagrangean lower bound program which is feasible for the original problem automatically satisfies both 1 and 2 above and so is optimal.

(3) If the solution to LLBP (for all possible multiplier (λ) values) is unchanged by replacing the integrality constraint $[x \in (0,1)]$ in LLBP by its linear relaxation $[0 \leq x \leq 1]$ then the Lagrangean relaxation/Lagrangean lower bound program is said to have the *integrality property*.

To illustrate this consider the Lagrangean relaxation of the SCP given above, which was:

$$\text{minimize} \quad \sum_{j=1}^{n} C_j x_j + \sum_{i=1}^{m} \lambda_i$$
$$\text{subject to} \quad x_j \in (0,1); \quad j = 1, \ldots, n$$

with solution

$$X_j = 1 \quad \text{if } C_j \leq 0$$
$$= 0 \quad \text{otherwise}$$

It is clear that replacing $x_j \in (0,1); j = 1, \ldots, n$ by $0 \leq x_j \leq 1; j = 1, \ldots, n$ leaves the solution unchanged.

Hence the Lagrangean relaxation of the SCP given above does have the integrality property.

(4) If the Lagrangean relaxation has the integrality property then the maximum lower bound attainable from LLBP is *equal* to the value of the linear programming relaxation of the original problem.

Hence for the Lagrangean relaxation of the SCP considered above, the maximum lower bound attainable from LLBP (i.e. the value of the Lagrangean dual program) is equal to the value of the LP relaxation of the original problem.

(5) If the Lagrangean relaxation does not have the integrality property then the maximum lower bound attainable from LLBP is *greater than (or equal to)* the value of the linear programming relaxation of the original problem.

(6) If a problem has a number of different sets of constraints then there are a number of possible Lagrangean relaxations. This is the strategic choice issue mentioned above, namely:

Which set (or sets) of constraints should we choose to relax?

The integrality property defined above provides a key mechanism to assist us in choosing the Lagrangean relaxation to use. It gives an indication of how good the maximum (best) lower bound we can get from a particular relaxation is.

Suppose we ask the question 'Does a Lagrangean relaxation have the integrality property?'. Then we have:

Answer: yes $\Rightarrow$ maximum lower bound = LP relaxation solution

no $\Rightarrow$ maximum lower bound $\geq$ LP relaxation solution

So a Lagrangean relaxation without the integrality property is to be preferred as it offers the prospect of getting better lower bounds.

But lower bounds from Lagrangean relaxations are not free. Computational effort is needed to solve LLBP for each set of values for the multipliers and, as we shall see below, we may have to solve LLBP for many different sets of multipliers.

Essentially in choosing the Lagrangean relaxation to use we have to balance the computational effort involved in solving LLBP against the maximum lower bound theoretically attainable (note that we may not, computationally, actually attain this bound).

A structured way to approach this strategic choice issue of which Lagrangean relaxation to use is by means of a simple table.

Consider the original problem (P):

$$\begin{aligned} \text{minimize} \quad & cx \\ \text{subject to} \quad & Ax \geq b \\ & Bx \geq d \\ & x \in (0,1) \end{aligned}$$

then we can set up the table:

Relaxation number	Constraints $Ax \geq b$ Relax?	Constraints $Bx \geq d$ Relax?	Computational effort involved in solving LLBP Number of multipliers	Computational effort involved in solving LLBP Number of operations	Integrality property?
1	no	no	?	?	?
2	no	yes	?	?	?
3	yes	no	?	?	?
4	yes	yes	?	?	?

For each set of constraints we have a choice of whether to relax them or not. Hence if there are s sets of constraints there are 2^s possible relaxations (in problem (P) above for example we have 2 sets of constraints so $s = 2$ and there are $2^2 = 4$ possible relaxations).

Filling in the details for 'Computational effort involved in solving LLBP' and 'Integrality property?' (the question marks in the above table) depends upon examining each relaxation in mathematical detail. Specifically we need to find:

- the number of Lagrange multipliers needed—this affects the amount of computer memory needed;

- the number of arithmetic operations required to solve LLBP (e.g. $O(n^2)$)—this affects the amount of computer time needed;

- whether LLBP has the integrality property or not.

Whilst the above table might seem somewhat daunting (e.g. 4 sets of constraints lead to $2^4 = 16$ possible relaxations), in practice some relaxations are not worth considering:

1. relaxation 1 in the table can be ignored since it is just the original problem (no constraints relaxed);

2. we need only consider relaxations for which 'Computational effort involved in solving LLBP' involves a polynomial (or pseudo-polynomial) number of operations, i.e. relaxations which are themselves 'hard' problems (in the NP-complete sense) are of no (computational) advantage;

3. all relaxations which have the integrality property have the same maximum lower bound, so the particular relaxation (with the integrality property) which involves the least computational effort in solving LLBP would seem the best choice to make. In particular note here that the relaxation which involves relaxing all constraints has the integrality property.

After 1, 2 and 3 above we are left with (hopefully) a small number of relaxations which are deserving of further investigation (either theoretically or computationally).

One hint that may be useful to researchers is to include a column in the above table relating to previous work dealing with each relaxation. This is helpful both in seeing where previous work has been concentrated and in seeing where there might be a legitimate 'gap' in previous research work.

(7) In fact there are many more Lagrangean relaxations of a problem than might be apparent from the table above. To see this consider the original problem (P):

$$\begin{aligned} \text{minimize} \quad & cx \\ \text{subject to} \quad & Ax \geq b \\ & Bx \geq d \\ & x \in (0,1) \end{aligned}$$

and transform this into:

$$\text{minimize} \quad cx$$
$$\text{subject to} \quad Ax \geq b$$
$$y = x$$
$$By \geq d$$
$$x \in (0,1)$$
$$y \in (0,1)$$

Here we introduce an artificial zero-one variable y equal to x and replace x in one set of constraints by y. It is clear that the original problem (P) and this transformed problem are equivalent.

If we now relax the equality constraint linking y and x together by introducing a Lagrange multiplier vector λ (now unrestricted in sign) we get the Lagrangean lower bound program (LLBP):

$$\text{minimize} \quad cx + \lambda(x - y)$$
$$\text{subject to} \quad Ax \geq b$$
$$By \geq d$$
$$x \in (0,1)$$
$$y \in (0,1)$$

and it is clear that LLBP is *separable* into the sum of two programs:

$$\text{minimize} \quad (c + \lambda)x \quad \text{and} \quad \text{minimize} \quad -\lambda y$$
$$\text{subject to} \quad Ax \geq b \qquad\qquad \text{subject to} \quad By \geq d$$
$$x \in (0,1) \qquad\qquad\qquad\qquad y \in (0,1)$$

The sum of the solutions to these two programs (for any value of λ) provides a lower bound on the optimal solution to the original (integer) problem.

This technique of introducing a 'copy' of the variables and then relaxing the equality constraint linking variables is known as *Lagrangean decomposition*. Informally it is clear that we can examine Lagrangean decomposition systematically by:

- introducing different artificial variables for all but one constraint (that constraint being expressed in terms of the original variables);

- adding all possible equality linking constraints between these artificial variables and the original variables (e.g. if we have

three constraints involving original variables x then we have two artificial variables (y and z, say) and the set of possible equality linking constraints is $\{x = y, x = z, y = z\}$);

- considering the relaxation of all equality linking constraints in a Lagrangean fashion using a table such as shown above.

More formally if there are s sets of constraints then there are $s(s-1)/2$ equality linking constraints and hence $2^{s(s-1)/2}$ possible Lagrangean decompositions.

(8) It may be possible to add to LLBP constraints which would have been redundant (unnecessary) in the original integer (or mixed-integer) formulation of the problem, but which strengthen the lower bound obtained from LLBP (for any value of λ). For example, for the SCP:

$$\text{minimize} \quad \sum_{j=1}^{n} c_j x_j$$
$$\text{subject to} \quad \sum_{j=1}^{n} a_{ij} x_j \geq 1; \quad i = 1, \ldots, m$$
$$x_j \in (0,1); \quad j = 1, \ldots, n$$

since $c_j > 0$ $(j = 1, \ldots, n)$ it is clear that at least one, but no more than m, variables can take the value 1 in the optimal solution and so the constraint

$$1 \leq \sum_{j=1}^{n} x_j \leq m$$

is valid (but redundant in the original formulation).

Whereas the LLBP for the SCP was given before by:

$$\text{minimize} \quad \sum_{j=1}^{n} C_j x_j + \sum_{i=1}^{m} \lambda_i$$
$$\text{subject to} \quad x_j \in (0,1); \quad j = 1, \ldots, n$$

adding this constraint to LLBP gives:

$$\text{minimize} \quad \sum_{j=1}^{n} C_j x_j + \sum_{i=1}^{m} \lambda_i$$
$$\text{subject to} \quad 1 \leq \sum_{j=1}^{n} x_j \leq m$$
$$x_j \in (0,1); \quad j = 1, \ldots, n$$

It is clear that:

- the lower bound obtained from this new (strengthened) LLBP is always (for any set of (λ_i)) better ($\geq$) than the lower bound obtained from the original LLBP; and

- this new (strengthened) LLBP is computationally easy to solve: sort the variables x_j by increasing C_j order and set $x_j = 1$ for at least one, and at most m, variables in this list depending upon whether $C_j \leq 0$ or not.

6.4   Lagrangean Heuristics and Problem Reduction

The solution to LLBP can be used to construct feasible solutions to the original problem (a Lagrangean heuristic) and also to reduce the size of the problem that we need to consider (problem reduction). We deal with these two topics below.

6.4.1   Lagrangean heuristic

In a Lagrangean heuristic we take the solution to LLBP and attempt to convert it into a feasible solution for the original problem by suitable adjustment (if necessary). This feasible solution constitutes an upper bound on the optimal solution to the problem (cf. Figure 6.1). To illustrate the concept of a Lagrangean heuristic we will develop a Lagrangean heuristic for the SCP.

In the set covering problem all feasible solutions consist of a set of columns (x_j) which cover each row at least once. In the solution to LLBP for the SCP we have some X_j one and some X_j zero. This may result in some rows not being covered; plainly these rows need to be covered to constitute a feasible solution for the SCP.

Hence one possible (very simple) Lagrangean heuristic is to construct a feasible solution S to the original SCP in the following way:

1. set $S = \{j | X_j = 1; j = 1, \ldots, n\}$

2. for each row i which is uncovered (i.e. $\sum_{j \in S} a_{ij} X_j = 0$) add the column corresponding to $\min_{j=1,\ldots,n} \{c_j | a_{ij} = 1\}$ to S

3. S will now be a feasible solution to the original SCP of cost $\sum_{j \in S} c_j$.

Note that the key feature of a Lagrangean heuristic is that we are building upon the current solution to LLBP. The essential idea here

is that just as the solution value for LLBP gives us useful information (a lower bound on the optimal solution to the original problem) so the *structure* of the solution to LLBP (i.e. the value of the variables) may well be giving us useful information about the structure of the optimal solution.

To illustrate the Lagrangean heuristic given above consider the example LLBP solution of $X_1 = 1, X_2 = X_3 = X_4 = 0$ that we had before. Applying this Lagrangean heuristic to our example LLBP solution we get:

1. $S = \{1\}$

2. row 3 is the only uncovered row and the minimum cost column covering this row is column 2 so add column 2 to S

3. $S = \{1, 2\}$ is now a feasible solution to the original SCP of cost $c_1 + c_2 = 2 + 3 = 5$.

Fortuitously here our Lagrangean heuristic has actually found the optimal solution to the original problem. Obviously this may not happen in all cases. However each time we solve LLBP the Lagrangean heuristic has an opportunity to transform the solution to LLBP into a feasible solution for the original problem. If, as is common in practice (see below), we solve LLBP many times then the Lagrangean heuristic has many opportunities to transform the solution to LLBP into a feasible solution for the original problem.

Designing a Lagrangean heuristic for a particular LLBP is an art, the success of which is judged solely by computational performance, i.e. whether a particular Lagrangean heuristic gives good quality (near-optimal or optimal) solutions in a reasonable computation time.

Our experience, based upon applying Lagrangean heuristics to a number of different problems (e.g. see [3, 4, 5, 6, 7, 8, 9, 10]) has been that relatively simple Lagrangean heuristics can give good quality results.

6.4.2   Problem reduction

Problem reduction consists of fixing the values of certain variables in the problem, enabling a reduction in problem size to be achieved. For example for the SCP:

- fixing $x_k = 1$ for some k enables us to delete column k and all rows covered by column k from the problem (since once rows have been covered by some column they need no longer be considered);

- fixing $x_k = 0$ for some k enables us to delete column k from the problem.

Let: R_0 be the set of variables fixed to zero;
R_1 be the set of variables fixed to one.

The basic idea here is to make reductions such that combining the optimal solution to the reduced problem with:

$$x_k = 0; \quad \forall k \in R_0$$
$$x_k = 1; \quad \forall k \in R_1$$

gives a solution to the original problem which is *optimal* for the original problem: i.e. the optimal solution to the original problem can be found by combining the optimal solution to the reduced problem with the reduction information.

This is important because our aim is to develop an algorithm that guarantees to find the optimal solution to the (original) set covering problem. The essential reason for doing problem reduction is the *hope* that the reduced problem will be quicker to solve in terms of computation time.

In combinatorial optimization we are often dealing with large problems with an enormous number of potential solutions. For example the SCP has $O(2^n)$ potential solutions (some of which may be infeasible). We need to find the (possibly unique) optimal solution. To give an idea of the size of the task, a current state-of-the-art algorithm for the set covering problem has solved problems with n=3000. Readers can tax their own calculators in finding a value for 2^{3000}!

Informally, one can think of looking for the optimal solution to a combinatorial optimization problem as analogous to looking for a needle in a haystack. Our task is to search the haystack to find the needle. By making a reduction in problem size, thereby eliminating potential solutions, we are throwing away portions of the haystack, reducing the amount that we have to search, in the hope that this will mean finding the needle more quickly!

To see how problem reduction can be achieved let Z_{UB} be the value of the best feasible solution found (e.g. by a Lagrangean heuristic) for the SCP we are considering.

Fixing $x_k = 0$

Problem reduction of the first type, involving fixing $x_k = 0$, can be carried out as follows:

1. From the original SCP and LLBP it is clear that imposing the additional constraint that a particular column k must be in the optimal solution ($x_k = 1$) results in a lower bound of $Z_{LB} + C_k$ for each column k for which $X_k = 0$. (Hint: solve LLBP with the additional constraint that $x_k = 1$). ◇

2. Thus we can remove column k from the problem if:

   (a) $X_k = 0$; and

   (b) $Z_{LB} + C_k > Z_{UB}$

   since a column k with $Z_{LB} + C_k$ (the lower bound corresponding to an optimal solution containing column k) greater than Z_{UB} cannot be in the optimal solution.

Figure 6.2 illustrates the situation. In that figure we have two points on the value line. One, an upper bound, Z_{UB} and the other, a lower bound, associated with the solution $(Z_{LB}, (X_j))$ to LLBP. Currently the (unknown) optimal solution to the problem lies somewhere in the interval $[Z_{LB}, Z_{UB}]$ (possibly at one of the two end-points of the interval).

Suppose that column k were to be part of the optimal solution (i.e. $x_k = 1$), then this solution would have to lie in the interval $[Z_{LB} + C_k, Z_{UB}]$. But we know that $Z_{LB} + C_k > Z_{UB}$ so we have a contradiction; thus column k cannot be part of the optimal solution. It can therefore be removed and the optimal solution to the reduced problem will be the same as the optimal solution to the original problem.

To illustrate problem reduction, consider our example LLBP solution with $X_1 = 1, X_2 = X_3 = X_4 = 0$ and $Z_{LB} = 4.2$. Currently no variables have been eliminated by problem reduction so $R_0 = \emptyset$ and
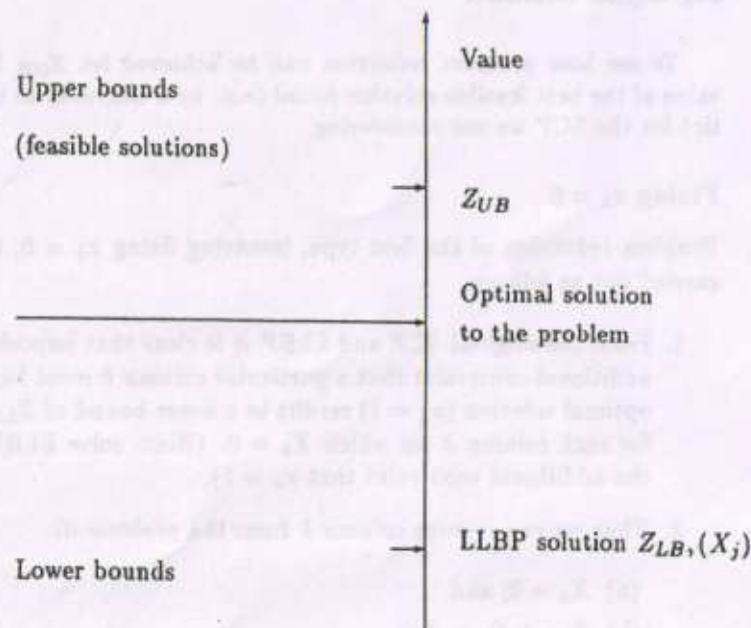
Value

Upper bounds

(feasible solutions)

Z_{UB}

Optimal solution

to the problem

LLBP solution $Z_{LB}, (X_j)$

Lower bounds

Figure 6.2: Upper and lower bounds

$R_1 = \emptyset$. For this solution we had $C_1 = -1.1, C_2 = 0.8, C_3 = 0.3$ and $C_4 = 1.2$.

We have found a feasible solution, via the Lagrangean heuristic, of value 5 so that $Z_{UB} = 5$. Hence:

$$Z_{LB} + C_k \quad = 4.2 + 0.8 = 5.0 \quad \text{for } k = 2 \text{ (as } X_2 = 0)$$
$$= 4.2 + 0.3 = 4.5 \quad \text{for } k = 3 \text{ (as } X_3 = 0)$$
$$= 4.2 + 1.2 = 5.4 \quad \text{for } k = 4 \text{ (as } X_4 = 0)$$

Therefore column 4 cannot be in the optimal solution as $X_4 = 0$ and $Z_{LB} + C_4 > Z_{UB}$. Hence $R_0 = R_0 \cup [x_4]$ and column 4 can be deleted from the problem.

Fixing $x_k = 1$

The second type of problem reduction, fixing $x_k = 1$, can be carried out as follows:

1. From the original SCP and LLBP, it is clear that imposing the additional constraint that a particular column k cannot be in the optimal solution ($x_k = 0$) results in a lower bound of $Z_{LB} - C_k$ if $X_k = 1$. (Hint: solve LLBP with the additional constraint that $x_k = 0$).

2. We can therefore fix column k in the solution (fix $x_k = 1$) if:

   (a) $X_k = 1$; and

   (b) $Z_{LB} - C_k > Z_{UB}$

   since a column k with $Z_{LB} - C_k$ (the lower bound corresponding to an optimal solution without column k) greater than Z_{UB} cannot be in the optimal solution.

The proof of the correctness of the above argument follows exactly the same lines as that for the first type of problem reduction, and will be left to the reader.

To illustrate problem reduction fixing $x_k = 1$ consider our example LLBP solution as shown above. We find that

$$Z_{LB} - C_k = 4.2 - (-1.1) = 5.3 \text{ for } k = 1 \text{ (as } X_1 = 1)$$

Therefore column 1 must be in the optimal solution as $X_1 = 1$ and $Z_{LB} - C_1 > Z_{UB}$. Hence $R_1 = R_1 \cup [x_1]$, and column 1 and the rows that it covers (rows 1 and 2) can be deleted from the problem.

Our experience, based upon applying problem reduction to a number of different problems (e.g. see [3, 4, 5, 6, 7, 8, 9, 10]), has been that problem reduction significantly improves the computational performance of any algorithm. For example the algorithm given in [7] for the SCP includes problem reduction and typically over 75% (often well over 75%) of the variables are eliminated by problem reduction.

6.4.3 Remarks

In the preceding two subsections we have tried to illustrate how a Lagrangean lower bound program solution can be used to yield useful information about the optimal solution, both in terms of a feasible solution (Lagrangean heuristic), and in terms of problem reduction.

One encouraging point to note from the numerical examples given above is that even though the lower bound ($Z_{LB} = 4.2$) was 16% away from the optimal solution of 5 we were able to gain important information, namely:

- we actually found the optimal solution through the Lagrangean heuristic; and

- we made significant problem reduction (deleting column 4 from the problem and identifying column 1 as being in the optimal solution).

Whilst we will not always be so lucky, the point is clear—useful information can still be deduced from LLBP solutions which, in lower bound terms, are quite far from the optimal solution.

6.5 Determination of Lagrange Multipliers

In the previous section we have seen how to apply Lagrangean relaxation to:

- generate a lower bound;

- generate an upper bound (corresponding to a feasible solution); and

- reduce the size of the problem.

In addition we gave some guidelines upon the strategic issue of deciding which, out of all possible relaxations, to choose.

In this section we deal with the tactical issue, namely given a particular relaxation (i.e. the strategic choice has been made), how to find numerical values for the multipliers.

There are two basic approaches to deciding values for the Lagrange multipliers (λ_i)- subgradient optimization, and multiplier adjustment. We deal with each in turn.

6.5.1 Subgradient optimization

Recall the original problem that we are attempting to solve:

$$\begin{aligned} \text{minimize} \quad & cx \\ \text{subject to} \quad & Ax \geq b \\ & Bx \geq d \\ & x \in (0,1) \end{aligned}$$

The Lagrangean lower bound program (LLBP) for this problem was:

$$\begin{aligned} \text{minimize} \quad & cx + \lambda(b - Ax) \\ \text{subject to} \quad & Bx \geq d \\ & x \in (0,1) \end{aligned}$$

the solution to which, for any $\lambda \geq 0$, gives a lower bound on the optimal solution to the original (integer) problem.

Subgradient optimization is an iterative procedure which, from an initial set of Lagrange multipliers, generates further multipliers in a systematic fashion. It can be viewed as a procedure which attempts to maximize the lower bound value obtained from LLBP (i.e. to solve the Lagrangean dual program—see above) by suitable choice of multipliers.

Switching from matrix notation to summation notation, so that the relaxed constraints are

$$\sum_{j=1}^{n} a_{ij} x_j \geq b_i; \quad i = 1, \ldots, m$$

the basic procedure is as follows:

1. Let π be a user-defined parameter satisfying $0 < \pi \leq 2$. Initialize Z_{UB} (e.g. from some heuristic for the problem). Decide upon an initial set (λ_i) of multipliers.

2. Solve LLBP with the current set of multipliers (λ_i), to get a solution (X_j) of value Z_{LB}.

3. Define *subgradients* G_i for the relaxed constraints, evaluated at the current solution, by:

$$G_i = b_i - \sum_{j=1}^{n} a_{ij} X_j; \quad i = 1, \ldots, m$$

4. Define a (scalar) step size T by

$$T = \frac{\pi(Z_{UB} - Z_{LB})}{\sum_{i=1}^{m} G_i^2}$$

This step size depends upon the gap between the current lower bound Z_{LB} and the upper bound Z_{UB} and the user defined parameter π (more of which below) with $\sum_{i=1}^{m} G_i^2$ being a scaling factor.

5. Update λ_i using

$$\lambda_i = \max(0, \lambda_i + TG_i); \quad i = 1, \ldots, m$$

and go to step 2 to resolve LLBP with this new set of multipliers.

As currently set out, this iterative procedure would never terminate, so we introduce a termination rule based either upon limiting the number of iterations that can be done, or upon the value of π where π is reduced during the course of the procedure. In the latter case, termination occurs when π is small—this will be discussed below.

We illustrate below one iteration of the subgradient optimization procedure for our example SCP.

1. Let $\pi = 2$.

Let $Z_{UB} = 6$ (e.g. suppose we have applied some heuristic for the SCP and have found a feasible solution $x_1 = x_3 = 1, x_2 = x_4 = 0$, of value 6).

Let $\lambda_1 = 1.5, \lambda_2 = 1.6, \lambda_3 = 2.2$, as before.

2. The solution to LLBP is $X_1 = 1, X_2 = X_3 = X_4 = 0$ with $Z_{LB} = 4.2$, as before.

3. The equations for the subgradients are:

$$\begin{aligned} G_1 &= (1 - X_1 - X_3) = 0 \\ G_2 &= (1 - X_1 - X_4) = 0 \\ G_3 &= (1 - X_2 - X_3 - X_4) = 1 \end{aligned}$$

4. The step size T is given by:

$$T = \frac{2(6 - 4.2)}{(0^2 + 0^2 + 1^2)} = 3.6$$

5. Updating λ_i gives:

$$\begin{aligned} \lambda_1 &= \max(0, 1.5 + 3.6(0)) = 1.5 \\ \lambda_2 &= \max(0, 1.6 + 3.6(0)) = 1.6 \\ \lambda_3 &= \max(0, 2.2 + 3.6(1)) = 5.8 \end{aligned}$$

Resolving LLBP with this new set of multipliers gives $X_1 = X_2 = X_3 = X_4 = 1$ with a new lower bound of $Z_{LB} = -0.7$.

Note here that, in this case, changing the multipliers has made the lower bound worse than before (before it was 4.2, much closer to the optimal solution of 5 than the new value of -0.7). This behaviour is common in subgradient optimization; we cannot expect, and do not observe, a continual improvement in the lower bound at each iteration. Indeed, as seen above, the lower bound can even become negative.

However, suppose that we let Z_{max} be the *maximum* lower bound found over all subgradient iterations, where initially $Z_{max} = -\infty$, and we update Z_{max} at each subgradient iteration using $Z_{max} = \max(Z_{max}, Z_{LB})$. What has been observed computationally, by many workers in the field, is that Z_{max} increases quite rapidly during the initial subgradient iterations with the rate of increase slowing as many iterations are performed. In fact, it is common for Z_{max} to approach very close to (or even attain) the maximum lower bound possible from the LLBP, i.e. for Z_{max} to approach very close to (or even attain) the value of the Lagrangean dual program.

Figure 6.3 illustrates the situation as we perform subgradient iterations. As shown in that figure, we plot the lower bound found at each subgradient iteration on the value line. The best (maximum) of these lower bounds is Z_{max}. This is the lower bound closest to the optimal solution.

Value

Upper bounds

(feasible solutions)

Z_{UB}

Optimal solution
to the problem
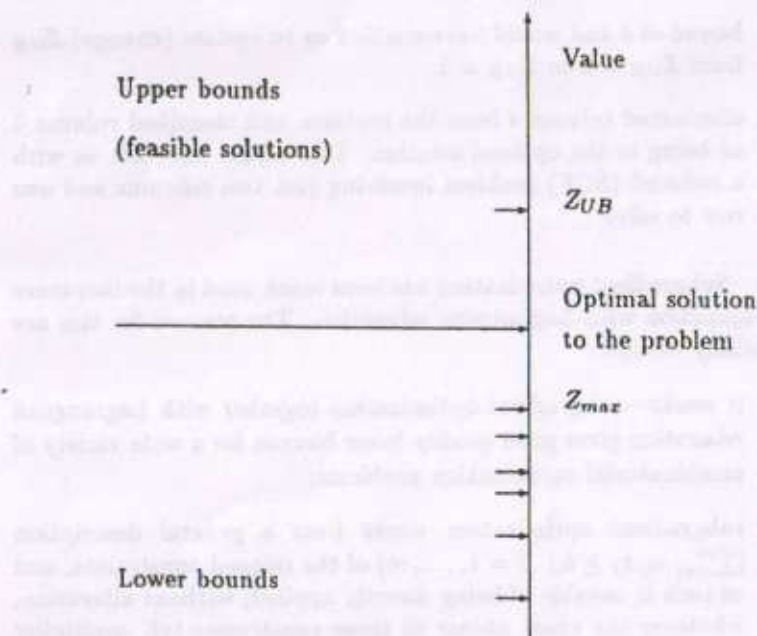
Z_{max}

Lower bounds

Figure 6.3: Subgradient optimization

6.5.2   Advanced subgradient optimization

Below we give a number of observations based upon our (extensive) experience with subgradient optimization.

(1)   It is clear that the subgradient optimization procedure can be terminated if we find that $Z_{max} = Z_{UB}$. In this case the value of the maximum lower bound (Z_{max}) coincides with the value of a feasible solution (Z_{UB}) and so Z_{UB} must be the optimal solution. As can be seen from Figure 6.3, the only place at which a lower bound and an upper bound can coincide on the value line is at the optimal solution.

(2)   Set $\pi = 2$ initially. If Z_{max} has not improved (i.e. increased) in the last N subgradient iterations with the current value of π then halve π. A value for N of 30 would seem to be reasonable, although

it might well be worthwhile experimenting (computationally) with different values of N, to see whether a different value generates significantly better results for the particular problem under consideration.

(3)   Unless terminated by (1) above, terminate the subgradient optimization procedure when π is small (e.g. $\pi \leq 0.005$). Often this means that many hundreds of subgradient iterations are done, but this is inevitable—if we want to generate a good quality lower bound (i.e. a value for Z_{max} which is close to the optimal solution) we cannot reasonably expect to do so in just a few subgradient iterations.

(4)   In the author's experience, how the Lagrange multipliers are initialized is not particularly important; how quickly the subgradient procedure terminates is relatively insensitive to the initial choice of multipliers, as is the quality of the final lower bound Z_{max} obtained.

(5)   It is helpful to replace Z_{UB} in the equation

$$T = \pi(Z_{UB} - Z_{LB})/\sum_{i=1}^{m} G_i^2$$

for the step size T by $1.05Z_{UB}$.

The reasoning here is that, ideally, we would like Z_{UB} and Z_{LB} to coincide ($Z_{UB} = Z_{LB}$ implies that we have found the optimal solution). As Z_{LB} approaches Z_{UB}, using the original equation for the step size implies that T gets smaller and smaller, so that we may experience many iterations as Z_{LB} creeps closer and closer to Z_{UB}. Replacing Z_{UB} by $1.05Z_{UB}$ (where the 1.05 is arbitrarily chosen) avoids this problem.

(6)   Suppose that $\lambda_i = 0$ and $G_i < 0$, then λ_i will remain zero after update. But a G_i^2 factor will been included in the equation for T. There seems little effective reason for this—after all we know that λ_i is not going to alter. Hence we have found it helpful to adjust the subgradients before calculating T using:

$$G_i = 0, \text{ if } \lambda_i = 0 \text{ and } G_i < 0$$

(7) It may happen that, at some subgradient iteration, $G_i = 0, \forall i$ (assuming that the adjustment referred to above has been carried out), and the subgradient procedure terminates. If this happens then:

1. if the current LLBP solution (X_j) is feasible for the original problem then it is the optimal solution for the original problem (i.e. it satisfies the conditions given earlier for the solution to LLBP to be the optimal solution to the original problem);

2. otherwise we simply have a lower bound Z_{max} on the optimal solution to the original problem.

(8) It is important to formulate the problem such that all equations which are to be relaxed are *scaled*, so that their right-hand sides have a maximum absolute value of one (or are zero).

This is because relaxing constraints with varying right-hand sides can lead to a constraint with a very large right-hand side dominating the step size expression for T.

(9) When relaxing an equality constraint then (as mentioned previously) λ_i can be positive or negative and the expression for updating λ_i becomes

$$\lambda_i = \lambda_i + TG_i$$

(10) Observant readers will have noted that nowhere in the subgradient procedure given above did we mention Lagrangean heuristics or problem reduction. Yet we know from our earlier discussion that both of these techniques yield useful information about the optimal solution, both in terms of an upper bound (Lagrangean heuristic) and in terms of problem reduction.

Plainly it is a simple matter to introduce both of these techniques into the subgradient procedure. Specifically, after solving LLBP, first apply the Lagrangean heuristic (in an attempt to update Z_{UB}), and then apply problem reduction.

For instance, had we done this in the subgradient example considered above we would have:

1. found from the Lagrangean heuristic a feasible solution of value 5, which is better than our (heuristically decided) initial upper

bound of 6 and would have enabled us to update (change) Z_{UB} from $Z_{UB} = 6$ to $Z_{UB} = 5$;

2. eliminated column 4 from the problem and identified column 1 as being in the optimal solution. This would have left us with a reduced (SCP) problem involving just two columns and one row to solve.

(11) Subgradient optimization has been much used in the literature in conjunction with Lagrangean relaxation. The reasons for this are essentially twofold:

- it works—subgradient optimization together with Lagrangean relaxation gives good quality lower bounds for a wide variety of combinatorial optimization problems;

- subgradient optimization works from a general description $[\sum_{j=1}^{n} a_{ij}x_j \geq b_i; \; i = 1, \ldots, m]$ of the relaxed constraints, and as such is capable of being directly applied, without alteration, whatever the exact nature of those constraints (cf. multiplier adjustment below).

6.5.3 Multiplier adjustment

Multiplier adjustment is simply a heuristic that:

- generates a starting set of Lagrange multipliers;

- attempts to improve them in some systematic way so as to generate an improved lower bound; and

- repeats the procedure if an improvement is made.

Often we simply change a single multiplier at each iteration, in contrast to subgradient optimization where we (potentially) change all multipliers at each iteration.

The advantages of multiplier adjustment are that

- it is usually computationally cheap; and

- it usually produces an increase (or at least no decrease) in the lower bound at each iteration.

The price we pay for this advantage is:

- the final lower bound obtained can be poor (i.e. worse than that obtained from subgradient optimization); and

- different problems require different multiplier adjustment algorithms (unlike subgradient optimization which is capable of being applied directly to many different problems).

Multiplier adjustment is sometimes called *Lagrangean dual ascent* as ← it can be viewed as an ascent procedure (i.e. a procedure with a monotonic improvement in the lower bound at each iteration) for the Lagrangean dual program.

As in the development of Lagrangean heuristics, developing multiplier adjustment algorithms is an art. However, *exactly* as for subgradient optimization above, where the equation for updating multipliers was:

$$\lambda_i = \max(0, \lambda_i + TG_i); \quad i = 1, \ldots, m$$

the direction in which we would like to change multipliers is clear:

- if $G_i < 0$ we would like to reduce λ_i;

- if $G_i = 0$ we leave λ_i unchanged;

- if $G_i > 0$ we would like to increase λ_i.

Hence one possible (very simple) multiplier adjustment algorithm for the SCP is:

1. solve LLBP with the current set of multipliers (λ_i) ;

2. choose any row i for which $G_i > 0$ (i.e. row i is uncovered in the current LLBP solution);

3. if row i is uncovered then it is easy to see from the relevant mathematics of LLBP that:

   (a) increasing λ_i will increase the lower bound obtained from LLBP; and

   (b) the maximum amount (δ) by which we can increase λ_i before the solution to LLBP changes is given by:

$$\delta = \min_{j=1,\ldots,n} (C_j | a_{ij} = 1)$$

   i.e. $\delta = \min(C_j)$ such that column j covers row i;

4. increase λ_i by δ and go to 1.

The above multiplier adjustment algorithm terminates when all rows are covered (i.e. $G_i \leq 0; \ \forall i$).

To illustrate this multiplier adjustment algorithm we shall apply it to our example SCP, starting from the multiplier values $\lambda_1 = 1.5, \lambda_2 = 1.6$ and $\lambda_3 = 2.2$ as before:

1. the solution to LLBP is $X_1 = 1, X_2 = X_3 = X_4 = 0, Z_{LB} = 4.2$ with $C_1 = -1.1, C_2 = 0.8, C_3 = 0.3, C_4 = 1.2$ and $G_1 = G_2 = 0, G_3 = 1$;

2. row 3 is uncovered as $G_3 > 0$;

3. columns 2, 3 and 4 cover row 3 so

$$\delta = \min(C_2, C_3, C_4) = 0.3;$$

4. we increase λ_3 by 0.3 to give a new set of multipliers $\lambda_1 = 1.5, \lambda_2 = 1.6, \lambda_3 = 2.5$.

Resolving LLBP with this new set of multipliers gives $X_1 = X_3 = 1, X_2 = X_4 = 0$ with a new lower bound of $Z_{LB} = 4.5$. As we would expect (from the manner in which we designed this multiplier adjustment algorithm), the lower bound has increased, and as all rows are now covered ($G_i \leq 0; \ \forall i$ in the LLBP solution associated with $Z_{LB} = 4.5$) the algorithm terminates.

Plainly we could have designed a better multiplier adjustment algorithm, for example investigating not just *increasing* λ_i as above, but also *reducing* λ_i. Discovering whether a particular multiplier adjustment algorithm gives good quality lower bounds at reasonable computational cost is a matter for computational experimentation.

We see here that, as previously remarked, in contrast to subgradient optimization where we simply apply a sequence of general formulae for subgradients, step size and Lagrange multiplier update, multiplier adjustment algorithm design is a much more creative (difficult!) process.

6.6 Dual Ascent

Dual ascent came to prominence with the work of Erlenkotter [11] and Bilde and Krarup [12] on the uncapacitated warehouse location problem which, computationally, was very successful.

6.6.1 Basic concepts

Consider the linear programming (LP) relaxation of any combinatorial optimization problem P (assumed to be a minimization problem, so that the LP relaxation is also a minimization problem).

The dual linear program associated with the LP relaxation is therefore a maximization problem. Hence:

$$\text{optimal P (integer) solution}$$
$$\geq$$
$$\text{LP relaxation solution}$$
$$=$$
$$\text{dual LP solution}$$
$$\geq$$
$$\text{any feasible solution for the dual LP}$$

Therefore any *heuristic* for the dual LP provides a way of generating a lower bound on the optimal integer solution of the original problem, since *any* dual feasible solution gives a lower bound on the optimal integer solution to the original problem.

Figure 6.4 illustrates the situation. In that figure we essentially have three regions:

1. upper bounds, the region above the optimal (integer) solution;

2. dual LP feasible solutions, the region below the LP relaxation solution; and

3. the gap, the region between the LP relaxation solution and the optimal (integer) solution.

Dual ascent consists therefore of simply thinking up some heuristic for generating feasible solutions to the dual of the LP relaxation of a problem.

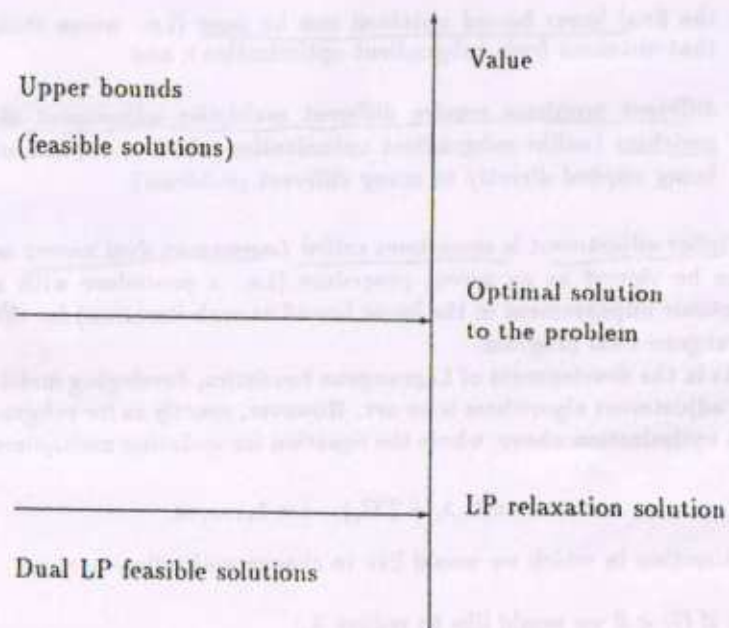We shall illustrate dual ascent with reference to the set covering problem. For the SCP the LP relaxation is:

Upper bounds

(feasible solutions)

Value

Optimal solution
to the problem

LP relaxation solution

Dual LP feasible solutions

Figure 6.4: Dual ascent

$$
\begin{array}{ll}
\text{minimize} & \sum_{j=1}^{n} c_j x_j \\
\text{subject to} & \sum_{j=1}^{n} a_{ij} x_j \geq 1; \quad i = 1, \ldots, m \\
& 0 \leq x_j \leq 1; \quad j = 1, \ldots, n
\end{array}
$$

As we have assumed (see above) that all costs c_j are strictly greater than zero this LP relaxation can be written as:

$$
\begin{array}{ll}
\text{minimize} & \sum_{j=1}^{n} c_j x_j \\
\text{subject to} & \sum_{j=1}^{n} a_{ij} x_j \geq 1; \quad i = 1, \ldots, m \\
& 0 \leq x_j; \quad j = 1, \ldots, n
\end{array}
$$

and the dual LP is:

$$
\begin{array}{ll}
\text{maximize} & \sum_{i=1}^{m} u_i \\
\text{subject to} & \sum_{i=1}^{m} u_i a_{ij} \leq c_j; \quad j = 1, \ldots, n \\
& u_i \geq 0; \quad i = 1, \ldots, m
\end{array}
$$

In order to illustrate dual ascent we will develop a dual ascent algorithm for our example SCP.

6.6.2 Example dual ascent algorithm

Considering the dual LP given above one possible (very simple) dual ascent algorithm is:

1. set $u_i = 0$; $\forall i$ (this is a feasible solution for the dual LP);

2. take each u_i $(i = 1, \ldots, m)$ in turn and increase it by as much as possible consistent with retaining feasibility.

A key point to note here is that often in a dual ascent algorithm we start from some dual feasible point and always retain dual feasibility throughout the algorithm. To illustrate the dual ascent algorithm given above we shall apply it to our example SCP, for which the dual LP is:

$$\begin{aligned}
\text{maximize} \quad & u_1 + u_2 + u_3 \\
\text{subject to} \quad & u_1 + u_2 \leq 2 \\
& u_3 \leq 3 \\
& u_1 + u_3 \leq 4 \\
& u_2 + u_3 \leq 5 \\
& u_1, u_2, u_3 \geq 0
\end{aligned}$$

Our simple dual ascent algorithm, as applied to this example, is therefore:

1. set $u_1 = u_2 = u_3 = 0$;

2. (a) after setting $u_2 = u_3 = 0$, the constraints involving u_1 are:

   $u_1 \leq 2$

   $u_1 \leq 4$

   so that u_1 can be increased to 2;

   (b) after setting $u_1 = 2, u_3 = 0$, the constraints involving u_2 are:

   $u_2 \leq 0$

   $u_2 \leq 5$

   so that u_2 cannot be increased;

(c) after setting $u_1 = 2, u_2 = 0$, the constraints involving u_3 are:

$u_3 \leq 3$

$u_3 \leq 2$

$u_3 \leq 5$

so u_3 can be increased to 2.

Hence we have a final solution of $u_1 = 2, u_2 = 0$ and $u_3 = 2$ which is a dual feasible solution and gives a lower bound of $u_1 + u_2 + u_3 = 4$.

Plainly we could have designed a better dual ascent algorithm, for example investigating not just increasing u_i as above, but also reducing u_i (thereby enabling us to increase other u_js). Discovering whether a particular dual ascent algorithm gives good quality lower bounds at reasonable computational cost is a matter for computational experimentation.

6.6.3 Connections

Consider the two techniques for generating lower bounds that we have given above, namely:

- Lagrangean relaxation (with the multipliers being decided by subgradient optimization or multiplier adjustment); and

- dual ascent, i.e. the heuristic solution of the dual of the LP relaxation of the problem.

Can we establish any connection between these two techniques? In fact we can by considering the question:

Is there any relationship between dual variables and Lagrange multipliers?

Recall here that we mentioned before that if a Lagrangean lower bound program (LLBP) had the *integrality property* then the maximum lower bound attainable from LLBP is equal to the value of the LP relaxation of the original problem.

However, it can also be shown that the values of the Lagrange multipliers that maximize the lower bound obtained from LLBP are given by the optimal values for the dual variables in the solution of the LP relaxation of the original problem.

In other words *if* the Lagrangean relaxation has the integrality property *then* the optimal Lagrange multipliers and the optimal dual variables are the same. This immediately implies that the maximum lower bound attainable from any Lagrangean relaxation with the integrality property is equal to the maximum lower bound attainable from any dual ascent algorithm for the problem.

6.6.4 Comparisons

Which of the three techniques—subgradient optimization, multiplier adjustment or dual ascent—should we use for obtaining lower bounds?

In the author's experience, subgradient optimization has always appeared to give very good lower bounds, in particular lower bounds often close to the optimal integer solution (so presumably also close to the maximum theoretically obtainable from the Lagrangean relaxation). Hence the author has never been very keen on multiplier adjustment methods although they appear useful for some problems, e.g. the generalized assignment problem [13, 14]. The only time the author tried dual ascent (for the *p*-median problem), it was a miserable failure!

There is a deeper point here. Techniques such as subgradient optimization, multiplier adjustment and dual ascent are *potentially* of wide applicability, i.e. they can be applied to a wide range of problems. However these techniques may *fail* computationally when applied across a wide range of problems, and may only be successful (possibly outstandingly successful) on one or two problems.

Based on this point then, to choose between these three lower bound techniques, the author's advice would be:

- subgradient optimization will nearly always work;

- multiplier adjustment may work;

- dual ascent will probably not work.

This advice is supported by evidence from the literature. Looking at the 'best' (most successful) optimal algorithm for a number of different problems, the reader will find that it is usually based on subgradient optimization and only occasionally based on multiplier adjustment or dual ascent.

6.6.5 Conclusions

We have now presented sufficient material for the diligent reader to be able to take a problem and develop lower bounds for it via Lagrangean relaxation. Applying subgradient optimization or multiplier adjustment will hopefully yield a good (close to integer optimal) lower bound.

Heuristic solutions

If we are simply interested in a heuristic solution for the problem we are considering, then the use of a Lagrangean heuristic throughout the subgradient procedure implies that at the end of that procedure:

- Z_{UB} is the value of the best feasible solution found;

- Z_{max} is the value of the best lower bound (on the optimal solution to the original problem) found; and

- the gap between Z_{UB} and Z_{max} (in percentage terms $100(Z_{UB} - Z_{max})/Z_{max}$) gives an indication of the quality of the solution (Z_{UB}) obtained from the Lagrangean heuristic.

In other words, *after* having solved the problem we find that we have a solution for which we have an associated *quality guarantee*. This fact is used extensively in many of the papers presented in the literature (see the discussion in section 6.8.).

However, we may be able to go further than simply accepting whatever quality guarantee figure we end up with after the subgradient procedure (note that we have no control or influence over this figure). This can be done by specifying, *before* we start to solve the problem, what quality guarantee we would like to achieve:

- suppose we are prepared to accept a solution within $\alpha\%$ of the optimum Z_{OPT}, i.e. we would accept a solution Z_{UB} with $(Z_{UB} - Z_{OPT}) \leq (\alpha/100)Z_{OPT}$, then

- *instead* of using Z_{UB} in any problem reduction test (for fixing variables to zero or one) use $Z_{UB}/[1 + (\alpha/100)]$.

If this is done then the subgradient procedure will terminate with either:

1. Z_{UB} proved to be within $\alpha\%$ of optimal, so that we will have achieved our desired (pre-specified) quality guarantee; or

2. a reduced problem, the optimal solution ZR_{OPT} of which has $\min(ZR_{OPT}, Z_{UB})$ within $\alpha\%$ of optimal. In this case we will not have achieved our pre-specified quality guarantee of $\alpha\%$ but will instead have a solution Z_{UB} with an associated quality guarantee of $100(Z_{UB} - Z_{max})/Z_{max}$ $(> \alpha\%)$.

Optimal solutions

If we are interested in finding the optimal solution to the problem we are considering then:

- we may already have found the optimal solution $(Z_{max} = Z_{UB})$; but

- if not we need to resort to a tree search (branch-and-bound) procedure to resolve the problem.

Tree search procedures are of course not heuristics—at least not in the sense in which we use the term in this book. (As has been pointed out in Chapter 1, some authors do use the word 'heuristic' for such methods, but it is not the consensus usage.) Nevertheless, for the sake of completeness, and for the benefit of readers who may wish to prove optimality in a particular situation, we consider tree search procedures below.

6.7   Tree Search

There are a number of different (and difficult) decisions that must be faced in designing a tree search procedure for a particular problem. In this section we concentrate upon designing a tree search procedure using a lower bound based upon Lagrangean relaxation and subgradient optimization. However, much of what is given here is equally applicable to tree search procedures using lower bounds based upon linear programming relaxation, dual ascent or multiplier adjustment.

6.7.1   Tree structure

In the author's experience, a binary tree search has always worked in every case on which it was tried.

6.7.2   Branching node

The author always uses a depth-first strategy (i.e. branch from the last available node in the tree). Again, this has always worked in all cases tried. The reader would only be recommended to try a more complex tree search if in complete despair at the results from a binary depth-first tree search.

6.7.3   Backtracking

We can backtrack from a tree node if either:

- the tree node is infeasible; or

- the lower bound at the tree node exceeds the upper bound.

Note here that the fact that the solution to the Lagrangean lower bound program (LLBP) at a particular tree node is a feasible solution to the original problem is *not* a condition for backtracking (see the fallacy mentioned above in section 6.3.5).

6.7.4   Forward branching rule

The author has always used a simple rule for forward branching based upon choosing a single variable and setting that variable equal to one (assuming here that we are dealing with a mathematical program involving zero-one variables).

Hence in a tree search procedure based upon a linear programming (LP) bound, for example, a reasonable (and simple) rule to use is: choose the variable that has the largest fractional part in the LP solution at the current tree node and branch by setting that variable to one.

As we are considering a tree search based upon Lagrangean relaxation, things are more complicated than this. Recall the LLBP:

$$\begin{aligned}
\text{minimize} \quad & cx + \lambda(b - Ax) \\
& = (c - \lambda A)x + \lambda b \\
\text{subject to} \quad & Bx \geq d \\
& x \in (0,1)
\end{aligned}$$

In a tree search procedure based upon a Lagrangean relaxation bound a reasonable rule to use is as follows:

1. Let (λ, X) represent the multipliers and variable values associated with the best lower bound found at the tree node from which we are branching.

2. Consider the vector of values $\lambda(b - AX) = \lambda G$ and choose the constraint (i, say) with the maximum (absolute) value in this vector.

   As discussed above, if X is feasible and satisfies $\lambda(b - AX) = 0$ then it is the optimal solution to the problem. Choosing the constraint with the maximum (absolute) value in $\lambda(b - AX)$ and then, by branching, attempting to satisfy that constraint (thereby reducing the contribution of constraint i to $\lambda(b - AX)$, i.e. attempting to make $\lambda(b - AX)$ zero) would seem logical.

3. Define:

$$S = \{j \mid x_j \text{ appears in constraint } i, X_j = 1\};$$

   if $S = \emptyset$ (i.e. no variables in constraint i are in the current LLBP solution X_j) then define:

$$S = \{j \mid x_j \text{ appears in constraint } i, X_j = 0\}$$

   Choose the variable to set to one that has the minimum LLBP objective function coefficient ($C = c - \lambda A$) out of those variables in S (i.e. the one that is most 'likely' to be one in an optimal completion of the current tree node).

Obviously, when choosing a variable to branch on at a particular tree node we exclude from consideration any variables already fixed to zero or one (either by explicit branching or by problem reduction at previous tree nodes).

When programming a tree search based upon Lagrangean relaxation, the author's advice would be to try the above branching rule,

and only investigate different branching rules if the computational results from the above rule are not satisfactory.

To illustrate the above branching rule, consider our example LLBP solution as given earlier with $Z_{LB} = 4.2$. Which variable would we choose to branch on?

1. We have $\lambda_1 = 1.5, \lambda_2 = 1.6, \lambda_3 = 2.2, X_1 = 1, X_2 = X_3 = X_4 = 0$. We also know that $C_1 = -1.1, C_2 = 0.8, C_3 = 0.3, C_4 = 1.2$ from above.

2. Then:

$$\begin{aligned}
\lambda_1(1 - X_1 - X_3) &= 0 \\
\lambda_2(1 - X_1 - X_4) &= 0 \\
\lambda_3(1 - X_2 - X_3 - X_4) &= 2.2
\end{aligned}$$

   Hence we would choose constraint 3. Note that this constraint corresponds to an uncovered row in the current LLBP solution ($X_1 = 1, X_2 = X_3 = X_4 = 0$).

3. $S = \{j \mid x_j \text{ appears in constraint } 3, X_j = 1\}$ gives $S = \emptyset$ since only columns (variables) 2, 3 and 4 appear in constraint 3 and they are all currently zero.

   Hence $S = \{j \mid x_j \text{ appears in constraint } 3, X_j = 0\}$ which gives $S = \{2, 3, 4\}$ and these variables have LLBP objective function coefficients of $C_2 = 0.8, C_3 = 0.3, C_4 = 1.2$. The minimum of these values is $C_3 = 0.3$, so we would choose x_3 to set to one in the forward branch.

6.7.5   Lower bound computation

Initial tree node

At the initial tree node (before branching), it is the author's experience that it is well worth while expending computational effort in doing many (of the order of hundreds) iterations.

This gives the Lagrangean heuristic a chance to find a good feasible solution, the problem reduction tests a chance to reduce the size of the problem and the subgradient procedure the chance to find a good lower bound. Given that hard problem instances, by definition, require many tree nodes for solution, the time spent at the initial tree

node soon becomes a small fraction of the overall computation time for such problems.

Other tree nodes

One option is to simply repeat, at each tree node, *exactly* the same procedure as at the initial tree node. The disadvantage of this is that it can make the lower bound computation at each tree node rather expensive. Essentially we have to balance the computational effort involved in calculating the lower bound at each tree node against the potential saving in tree nodes resulting from improving that lower bound.

Nowadays the author tends to use the strategy of:

1. initialising the Lagrange multipliers at each tree node with the set associated with the best (maximum) lower bound found at the predecessor tree node;

2. carrying out 30 subgradient iterations, with π (the user-defined parameter satisfying $0 < \pi \leq 2$) being halved every 5 (or 10) iterations at tree nodes associated with forward branching;

3. doubling both these figures (to 60 and 10 (or 20) respectively) for tree nodes associated with backtracking.

The logic for step 3 above is that in backtracking we are explicitly setting to zero in the tree a variable which, when we carried out the forward branch, was a good candidate for taking the value one (cf. the rule for forward branching given above). It seems reasonable therefore to do more iterations at the backtrack tree node in an attempt to achieve a lower bound sufficient to eliminate the node from further consideration.

6.7.6 Computer programs

Clearly a number of readers of this chapter and book are likely to be doctoral students who are not only gaining additional knowledge, but may also decide to apply a number of the techniques given in this chapter to a particular research problem. For this reason we feel that some guidance/help should be given about sources of available

computer programs. The help that we can offer the reader falls into two categories:

- solving the Lagrangean relaxation;

- problem reduction.

We deal with each in turn.

Lagrangean relaxation

Recall here that one of the guidelines for choosing a Lagrangean relaxation was that the relaxed problem (the Lagrangean lower bound program) was, computationally, easy to solve. Often the LLBP is a standard problem, for example a knapsack problem.

Generally it is best (if at all possible) to use commercial/publically available software to solve these standard problems rather than writing one's own software (software development is both time-consuming and frustrating!). An excellent source of information on available commercial software is contained in the advertisements/software reviews in *OR/MS Today* (the magazine associated with membership of the Operations Research Society of America).

Table 6.1 gives brief details of additional useful software.

Problem reduction

Note here that, in any program code, it is often difficult to alter data structures to reflect explicitly the fact that variables have been eliminated from the problem (either by being fixed to zero or by being fixed to one). There are some specific ways to avoid altering data structures:

- If a variable x_k is fixed to zero then simply alter its cost c_k from its original value to infinity (in computer terms a very large number). This will ensure that the variable never appears in a solution to the LLBP (nor, presumably, in any solution found by the Lagrangean heuristic).

- If a variable x_k is fixed to one then:

Table 6.1: Useful software

Linear/integer programming

There are many linear programming packages available, often with extensions to allow mixed-integer (or pure integer) problems to be solved by LP based tree search. One worth noting is the Marsten XMP code which is unusual in that one has complete access to the FORTRAN source (contact XMP Software, PO Box 58119, Tucson AZ 85732, USA).

Knapsack problem

The book by Martello and Toth, *Knapsack Problems: Algorithms and Computer Implementations* (Wiley, 1990) contains a number of codes.

Generalized assignment problem

A code is available from Guignard and Rosenwein [14].

Assignment/transportation/network flow problems

A code is available from Bertsekas and Tseng (Operations Research **36** (1988), 93-114).

Miscellaneous

A number of codes are available from ORSEP (Operations Research Software Exchange Programme) - see issues of the European Journal of Operational Research.
Packages (such as QSB+, STORM), capable of solving relatively small problems on IBM-PCs, are readily available, e.g. see the review by Beasley (Journal of the Operational Research Society **39** (1988), 487-509.
There are several books which contain useful codes e.g. Syslo, Deo and Kowalik, *Discrete Optimization Algorithms: with Pascal programs* (Prentice-Hall, 1983).
The NAG library (contact NAG, Wilkinson House, Jordan Hill Road, Oxford OX2 8DR, England) contains a number of useful codes.

---

1. any relaxed constraint i which is always satisfied because $x_k = 1$ can be:

   (a) neglected in LLBP simply by always setting $\lambda_i = 0$;

   (b) explicitly neglected in any Lagrangean heuristic;

2. accumulate a total C_{add}, which is the total cost of variables identified as being in the solution, and:

   (a) alter the cost c_k of x_k from its original value to a value v_k, say, sufficient to ensure that x_k will always automatically appear in the solution to LLBP (for the SCP example given above $v_k = 0$ would be appropriate);

   (b) add C_{add} minus the sum of all the v_k to the solution value given by LLBP to get a valid lower bound;

   (c) ensure that all x_k are included in the Lagrangean heuristic solution procedure.

Our experience has been that, after a significant amount of reduction has been achieved, e.g. 10% of variables fixed at zero or one, it is computationally worthwhile to perform 'house cleaning' operations, i.e. explicitly to alter data structures to reflect the fact that variables have been eliminated from the problem (either by being fixed to zero or by being fixed to one).

The reasoning here is clear: the elimination of just one variable is probably not sufficient to justify the computational overhead of altering data structures, whereas after a significant amount of reduction it is computationally worthwhile altering data structures. Essentially we have to balance the computational effort involved in altering the data structure against the possible reduction in overall computational effort in future iterations resulting from an altered data structure.

In fact, although we have not mentioned it above, this balance between the computational effort involved in some procedure and the future benefits (computational reduction) from the procedure is a key consideration in effective algorithm design.

For example, should we carry out problem reduction (a computational procedure) at each subgradient iteration or only at selected subgradient iterations (e.g. at iterations when Z_{max} increases)? Answering such a question is a matter of specific computational experimentation, i.e. trying different strategies and seeing which is best.

However it is important to be aware that such a question needs to be addressed.

6.8   Applications

As mentioned at the very beginning of this chapter much of what is contained in it is historical, and has been known for several years. To enable the reader to get some insight into current up-to-date work involving Lagrangean relaxation we have, in this section, adopted the strategy of:

- selecting calendar year 1991 (the most recent complete calendar year at the time of writing);

- selecting a number of journals; and

- reviewing all papers in those journals in our selected year which use Lagrangean relaxation.

This provides *systematic* insight into the range of problems to which Lagrangean relaxation is being applied.

This approach is not common in chapters of this kind. What is usually done is for the author to present a few 'selected highlights', i.e. papers drawn from a number of years that are meant to be especially significant. We have adopted a different approach because we believe that Lagrangean relaxation has wide applicability, and that a systematic survey (rather than 'selected highlights') provides *evidence* for such a belief [1].

By the end of this chapter readers will be able to judge for themselves whether Lagrangean relaxation has wide applicability or not.

6.8.1   Journals

The journals that we have selected for systematic review are:

- *Operations Research*;

---

[1] *Editorial footnote*: Readers will notice that the alternative strategy is indeed that adopted in the main by other contributors to this book. It is perhaps because Lagrangean relaxation is (relatively speaking) so much more developed than the other techniques presented that Beasley's approach is possible.

- *Management Science*;

- *European Journal of Operational Research*.

These journals are some of the most prestigious and most cited journals in the field of Operations Research/Management Science.

Operations Research

In 1991 *Operations Research* published 86 papers (of assorted lengths) spread over 1026 journal pages. Of these, 6 papers, spread over 76 pages, (7.0% and 7.4% respectively) involved Lagrangean relaxation. These papers are dealt with below.

- Ahmadi and Matsuo [15] present a paper dealing with the line segmentation problem. This is the problem of allocating the machines in a multi-stage production line to a number of different 'families' of items.

  They present a quadratic integer programming formulation of the problem and use Lagrangean relaxation, subgradient optimization and a Lagrangean heuristic. Computational results are presented for problems based on real-world data for large-scale circuit board manufacturing.

- Balas and Saltzman [16] present a paper dealing with the three-index assignment problem. This is a generalization of the well-known (two-index) assignment problem (involving variables x_{ij} indicating whether i and j are assigned together) to a three-index problem (involving variables x_{ijk} indicating whether i, j and k are assigned together).

  They present a zero-one formulation of the problem and use Lagrangean relaxation, subgradient optimization, dual ascent and a Lagrangean heuristic in conjunction with a tree search procedure. Computational results are given for randomly generated problems.

- Shulman [17] presents a paper dealing with the dynamic capacitated plant location problem. This is the problem of deciding the size, location and opening period of plants so as to service customers at minimum cost over a specified time horizon.

He presents a mixed-integer programming formulation of the problem and uses Lagrangean relaxation, subgradient optimization and Lagrangean heuristics. Computational results are presented for a number of randomly generated problems.

- Altinkemer and Gavish [18] present a paper dealing with the delivery problem. This is the problem of deciding minimum cost routes for a fleet of delivery vehicles (of known capacity) which are based at a central depot and have to visit a set of customers.

They present a zero-one formulation of the problem and give a number of heuristics for the problem. A lower bound, based upon Lagrangean relaxation and subgradient optimization, is used to evaluate the quality of the heuristic solutions. Computational results are presented for a number of problems (both taken from the literature and randomly generated).

- Noon and Bean [19] present a paper dealing with the asymmetric generalized travelling salesman problem. This is a generalization of the well-known travelling salesman problem in which there are sets of cities, and the tour must include exactly one city from each of the (mutually exclusive) sets.

They present a zero-one formulation of the problem and use Lagrangean relaxation, subgradient optimization, problem reduction and a Lagrangean heuristic in conjunction with a tree search procedure. Computational results are presented for a number of randomly generated problems.

- Ahmadi and Tang [20] present a paper dealing with the operation partitioning problem. This is the problem of assigning operations to machines so as to minimize the total movement of jobs between machines.

They present two zero-one formulations of the problem and use Lagrangean relaxation, subgradient optimization and Lagrangean heuristics. Computational results are presented for a number of randomly generated problems.

Management Science

In 1991 *Management Science* published 115 papers spread over 1652 journal pages. Of these 6 papers, spread over 109 journal pages, (5.2% and 6.6% respectively) involved Lagrangean relaxation. These papers are dealt with below.

- Pirkul and Schilling [21] present a paper dealing with the capacitated maximal covering problem. This is the problem of locating a certain number of facilities to cover (service) selected demand points, where facilities are both limited in size and in the total demand that they can service. This problem is related to locating emergency service facilities, such as fire and ambulance stations.

They use a zero-one formulation of the problem and apply Lagrangean relaxation with subgradient optimization and a Lagrangean heuristic. Computational results are presented for a large number of problems, including a 'real world' problem involving locating ten fire stations in an area of 85 square miles involving 625 demand points.

- Campbell and Mabert [22] present a paper dealing with planning production of a number of items on a single machine. The key feature of this problem is that production is cyclical, i.e. for each item the time between each lot (batch) being produced is a constant.

They present a mixed-integer programming formulation of the problem and use Lagrangean relaxation, subgradient optimization and a Lagrangean heuristic. Computational results are presented for a number of problems, including some based on data from the Ford Motor company.

- Gavish and Pirkul [23] present a paper dealing with the multi-resource generalized assignment problem. This is the problem of minimising the total cost involved in assigning tasks to agents, where associated with each possible task/agent assignment is a vector of resources and there are limits on the resources available to each agent.

They present a zero-one formulation of the problem and apply Lagrangean relaxation, subgradient optimization, Lagrangean

heuristics and problem reduction in conjunction with a tree search procedure. Computational results are presented for a number of randomly generated problems.

- Domich et al. [24] present a paper dealing with the problem of locating offices for the American Internal Revenue Service (IRS). This problem can be formulated as an uncapacitated fixed-charge location-allocation problem.

They use a zero-one formulation of the problem and apply Lagrangean relaxation with subgradient optimization and a Lagrangean heuristic. Computational results are presented for real-world data based upon locating IRS offices in Florida.

- Bard and Bejjani [25] present a paper dealing with the problem of leasing telecommunication lines so as to achieve a desired level of service at minimum cost.

They present a dynamic programming formulation of the problem which involves solving a number of general integer programs. These integer programs are solved using Lagrangean relaxation, subgradient optimization and a Lagrangean heuristic. Computational results are presented for real-world data involving a telecommunications company based in Texas.

- Drexl [26] presents a paper dealing with resource constrained project scheduling. This is the problem of minimising the cost of performing a set of jobs subject to job precedence constraints and resource restrictions.

He presents a zero-one formulation of the problem and uses a number of bounds (including a bound based on Lagrangean relaxation and multiplier adjustment) in conjunction with a tree search procedure. Computational results are presented for a number of problems (both taken from the literature and randomly generated).

European Journal of Operational Research

In 1991 the European Journal of Operational Research published 207 papers spread over 2342 journal pages. Of these 9 papers, spread over 108 pages, (4.3% and 4.6% respectively) involved Lagrangean relaxation. These papers are dealt with below.

- Cornuejols et al. [27] present a paper dealing with the capacitated plant location problem. This is the problem of deciding the number, and location, of plants to service customer demands, at minimum cost, where there is a limit on the total demand that can be serviced from each plant.

They present a mixed-integer programming formulation of the problem and consider a number of different Lagrangean relaxations. They use subgradient optimization and a Lagrangean heuristic. Computational results are presented for a number of problems (both taken from the literature and randomly generated).

- Aboudi et al. [28] present a paper dealing with the well-known assignment problem but with additional constraints involving equality between certain solution variables.

They present a zero-one formulation of the problem and use Lagrangean relaxation and a procedure for generating valid inequalities violated by Lagrangean relaxation solutions. No computational results are given (only a small numerical example).

- Sharma [29] presents a paper dealing with a problem involving fertiliser distribution in India. This is a problem of deciding how much fertiliser to produce at each manufacturing plant, which warehouses to use to store it (and their sizes), and how to distribute fertiliser to customers over a specified time horizon.

He presents a mixed-integer programming formulation of the problem and uses Lagrangean relaxation, subgradient optimization and a Lagrangean heuristic. Computational results are presented for problems based upon real-world data.

- Deckro et al. [30] present a paper dealing with the multi-project resource constrained scheduling problem. This is the problem of minimizing the cost of performing a set of jobs (spread across a number of projects) subject to job precedence constraints and resource restrictions.

They present a zero-one formulation of the problem and use Lagrangean relaxation. Computational results are presented for a number of randomly generated problems.

- Lozano et al. [31] present a paper dealing with the single level capacitated dynamic lot-sizing problem. This is the problem of determining the quantities and timing of production batches in order to meet customer demands at minimum cost over a specified time horizon.

They present a mixed-integer programming formulation of the problem and use Lagrangean relaxation. Computational results are presented for a number of problems (both taken from the literature and randomly generated).

- Wright and von Lanzenauer [32] present a paper dealing with the fixed-charge problem. This is a linear programming problem with discontinuities in the objective function resulting from fixed charges associated with non-zero variable values.

They present a mixed-integer programming formulation of the problem and use Lagrangean relaxation, multiplier adjustment and a Lagrangean heuristic. Computational results are presented for a number of problems taken from the literature.

- Current and Pirkul [33] present a paper dealing with the problem of designing a minimum cost two-level network, consisting of a primary path between an origin node and a destination node, with all nodes not on the path being connected to the path. In addition trans-shipment facilities must be located on the primary path.

They present a zero-one formulation of the problem and use Lagrangean relaxation, subgradient optimization and Lagrangean heuristics. Computational results are presented for a number of randomly generated problems.

- Barcelo et al. [34] present a paper dealing with the capacitated plant location problem. This is the problem of deciding the number, and location, of plants to service customer demands, at minimum cost, where there is a limit on the total demand that can be serviced from each plant.

They present a mixed-integer programming formulation of the problem and use Lagrangean relaxation, Lagrangean decomposition, subgradient optimization, a Lagrangean heuristic and

problem reduction in conjunction with a tree search procedure. Computational results are presented for a number of randomly generated problems.

- Millar and Gunn [35] present a paper dealing with trawler fleet dispatching. This is the problem of finding a minimum cost plan for using fishing trawlers in order to meet demand for various species of fish at processing plants over a specified time horizon.

They present a mixed-integer programming formulation of the problem and use Lagrangean relaxation and subgradient optimization. Computational results are presented for real-world data involving a large fish-processing company based in Canada.

6.8.2 Summary

Table 6.2: Summary of 1991 papers reviewed

Authors	Subgr. opt.	Mult. adj.	Dual ascent	Lagr. heur.	Prob. red.	Tree search
Ahmadi/Matsuo [15]	yes	-	-	yes	-	-
Balas/Saltzman [16]	yes	-	yes	yes	-	yes
Shulman [17]	yes	-	-	yes	-	-
Altinkemer/Gavish [18]	yes	-	-	-	-	-
Noon/Bean [19]	yes	-	-	yes	yes	yes
Ahmadi/Tang [20]	yes	-	-	yes	-	-
Pirkul/Schilling [21]	yes	-	-	yes	-	-
Campbell/Mabert [22]	yes	-	-	yes	-	-
Gavish/Pirkul [23]	yes	-	-	yes	yes	yes
Domich et al. [24]	yes	-	-	yes	-	-
Bard/Bejjani [25]	yes	-	-	yes	-	-
Drexl [26]	-	yes	-	-	-	yes
Cornuejols et al. [27]	yes	-	-	yes	-	-
Aboudi et al. [28]	-	-	-	-	-	-
Sharma [29]	yes	-	-	yes	-	-
Deckro et al. [30]	-	-	-	-	-	-
Lozano et al. [31]	-	-	-	-	-	-
Wright/von Lanzenauer [32]	-	yes	-	yes	-	-
Current/Pirkul [33]	yes	-	-	yes	-	-
Barcelo et al. [34]	yes	-	-	yes	yes	yes
Millar/Gunn [35]	yes	-	-	-	-	-

Table 6.2 provides a summary of the papers considered above. The popularity of subgradient optimization and Lagrangean heuristics is clear from this table. In particular note that a common theme nowadays is to use a Lagrangean heuristic to generate a feasible solution to a problem (thereby automatically obtaining an associated quality guarantee), without going to the computational expense of solving the problem optimally.

6.9   Conclusions

In this chapter we have discussed:

- Lagrangean relaxation
- Lagrangean heuristics
- problem reduction
- subgradient optimization
- multiplier adjustment
- dual ascent
- tree search

Plainly there are many possible techniques that can be brought to bear on combinatorial optimization problems in an attempt to develop successful algorithms. In this chapter we have outlined a number of the more popular techniques.

As mentioned before, developing algorithms which are computationally successful at solving combinatorial optimization problems is an art. However, the basic building blocks are clear—the types of techniques considered in this and other chapters of this book.

In many ways the work that goes on (all around the world) developing algorithms for the solution of combinatorial optimization problems is analogous to a large group of children playing with plastic bricks in an attempt to build a house. All children have the same set of bricks to play with but some are more successful than others at building a house. Some, indeed, enjoy the process more than others.

One of the appeals of solving combinatorial optimization problems, at least for this author, is that only a small amount of knowledge is required in order to join in the game with the other children and, through experience, one learns to build better houses with the building blocks at one's disposal.

Further references

The list of references on the following pages also includes additional reading as follows:

- for the set covering problem see [4, 7, 36, 38, 41];
- for Lagrangean relaxation see [39, 40, 43];
- for subgradient optimization see [37, 39, 40, 43, 44, 49];
- for multiplier adjustment see [39, 40, 45, 48];
- for dual ascent see [11, 12, 16, 42];
- for Lagrangean decomposition see [46, 47, 50, 51].

References

[1] M.Held and R.M.Karp (1970) The travelling-salesman problem and minimum spanning trees. *Ops.Res.*, **18**, 1138-1162.

[2] M.Held and R.M.Karp (1971) The travelling-salesman problem and minimum spanning trees: Part II. *Math.Prog.*, **1**, 6-25.

[3] J.E.Beasley (1985) A note on solving large *p*-median problems. *EJOR*, **21**, 270-273.

[4] J.E.Beasley (1987) An algorithm for set covering problems. *EJOR*, **31**, 85-93.

[5] J.E.Beasley (1988) An algorithm for solving large capacitated warehouse location problems. *EJOR*, **33**, 314-325.

[6] J.E.Beasley (1989) An SST-based algorithm for the Steiner problem in graphs. *Networks*, **19**, 1-16.

[7] J.E.Beasley (1990) A Lagrangean heuristic for set covering problems. *NRL*, **37**, 151-164.

[8] J.E.Beasley (1992) Lagrangean heuristics for location problems. *EJOR*, (to appear).

[9] N.Christofides and J.E.Beasley (1982) A tree search algorithm for the *p*-median problem. *EJOR*, **10**, 196-204.

[10] N.Christofides and J.E.Beasley (1983) Extensions to a Lagrangean relaxation approach for the capacitated warehouse location problem. *EJOR*, **12**, 19-28.

[11] D.Erlenkotter (1978) A dual-based procedure for uncapacitated facility location. *Ops.Res.*, **26**, 992-1009.

[12] O.Bilde and J.Krarup (1977) Sharp lower bounds and efficient algorithms for the simple plant location problem. *Annals of Discrete Math.*, **1**, 79-88.

[13] M.L.Fisher, R.Jaikumar and L.N.Van Wassenhove (1986) A multiplier adjustment method for the generalized assignment problem. *Man.Sci.*, **32**, 1095-1103.

[14] M.Guignard and M.B.Rosenwein (1989) An improved dual based algorithm for the generalized assignment problem. *Ops.Res.*, **37**, 658-663.

[15] R.H.Ahmadi and H.Matsuo (1991) The line segmentation problem. *Ops.Res.*, **39**, 42-55.

[16] E.Balas and M.J.Saltzman (1991) An algorithm for the three-index assignment problem. *Ops.Res.*, **39**, 150-161.

[17] A.Shulman (1991) An algorithm for solving dynamic capacitated plant location problems with discrete expansion sizes. *Ops.Res.*, **39**, 423-436.

[18] K.Altinkemer and B.Gavish (1991) Parallel savings-based heuristics for the delivery problem. *Ops.Res.*, **39**, 456-469.

[19] C.E.Noon and J.C.Bean (1991) A Lagrangean based approach for the asymmetric generalized travelling salesman problem. *Ops.Res.*, **39**, 623-632.

[20] R.H.Ahmadi and C.S.Tang (1991) An operation partitioning problem for automated assembly system design. *Ops.Res.*, **39**, 824-835.

[21] H.Pirkul and D.A.Schilling (1991) The maximal covering location problem with capacities on total workload. *Man.Sci.*, **37**, 233-248.

[22] G.M.Campbell and V.A.Mabert (1991) Cyclical schedules for capacitated lot sizing with dynamic demands. *Man.Sci.*, **37**, 409-427.

[23] B.Gavish and H.Pirkul (1991) Algorithms for the multi-resource generalized assignment problem. *Man.Sci.*, **37**, 695-713.

[24] P.D.Domich, K.L.Hoffman, R.H.F.Jackson and M.A.McClain (1991) Locating tax facilities: a graphics-based microcomputer optimization model. *Man.Sci.*, **37**, 960-979.

[25] J.F.Bard and W.A.Bejjani (1991) Designing telecommunications networks for the reseller market. *Man.Sci.*, **37**, 1125-1146.

[26] A.Drexl (1991) Scheduling of project networks by job assignment. *Man.Sci.*, **37**, 1590-1602.

[27] G.Cornuejols, R.Sridharan and J.M.Thizy (1991) A comparison of heuristics and relaxations for the capacitated plant location problem. *EJOR*, **50**, 280-297.

[28] R.Aboudi, A.Hallefjord and K.Jörnsten (1991) A facet generation and relaxation technique applied to an assignment problem with side constraints. *EJOR*, **50**, 335-344.

[29] R.R.K.Sharma (1991) Modelling a fertiliser distribution system. *EJOR*, **51**, 24-34.

[30] R.F.Deckro, E.P.Winkofsky, J.E.Hebert and R.Gagnon (1991) A decomposition approach to multi-project scheduling. *EJOR*, **51**, 110-118.

[31] S.Lozano, J.Larraneta and L.Onieva (1991) Primal-dual approach to the single level capacitated lot-sizing problem. *EJOR*, **51**, 354-366.

[32] D.Wright and C.H.von Lanzenauer (1991) COAL: a new heuristic approach for solving the fixed charge problem—computational results. *EJOR*, **52**, 235-246.

[33] J.Current and H.Pirkul (1991) The hierarchical network design problem with trans-shipment facilities. *EJOR*, **52**, 338-347.

[34] J.Barcelo, E.Fernandez and K.O.Jörnsten (1991) Computational results from a new Lagrangean relaxation algorithm for the capacitated plant location problem. *EJOR*, **53**, 38-45.

[35] H.H.Millar and E.A.Gunn (1991) Dispatching a fishing trawler fleet in the Canadian Atlantic groundfish industry. *EJOR*, **55**, 148-164.

[36] E.Balas and A.Ho (1980) Set covering algorithms using cutting planes, heuristics, and subgradient optimization: a computational study. *Math.Prog.*, **12**, 37-60.

[37] M.S.Bazaraa and J.J.Goode (1979) A survey of various tactics for generating Lagrangean multipliers in the context of Lagrangean duality. *EJOR*, **3**, 322-338.

[38] T.J.Chan and C.A.Yano (1992) A multiplier adjustment approach for the set partitioning problem. *Ops.Res.*, **40**, S40-S47.

[39] M.L.Fisher (1981) The Lagrangean relaxation method for solving integer programming problems. *Man.Sci.*, **27**, 1-18.

[40] M.L.Fisher (1985) An applications oriented guide to Lagrangean relaxation. *Interfaces*, **15**, 10-21.

[41] M.L.Fisher and P.Kedia (1990) Optimal solution of set covering/partitioning problems using dual heuristics. *Man.Sci.*, **36**, 674-688.

[42] R.D.Galvao (1980) A dual-bounded algorithm for the p-median problem. *Ops.Res.*, **28**, 1112-1121.

[43] A.M.Geoffrion (1974) Lagrangean relaxation for integer programming. *Math.Prog. Study*, **2**, 82-114.

[44] J.L.Goffin (1977) On the convergence rates of subgradient optimization methods. *Math.Prog.*, **13**, 329-347.

[45] M.Guignard (1988) A Lagrangean dual ascent algorithm for simple plant location problems. *EJOR*, **35**, 193-200.

[46] M.Guignard and S.Kim (1987) Lagrangean decomposition: a model yielding stronger Lagrangean bounds. *Math.Prog.*, **39**, 215-228.

[47] M.Guignard and S.Kim (1987) Lagrangean decomposition for integer programming: theory and applications. *RAIRO*, **21**, 307-323.

[48] M.Guignard and M.B.Rosenwein (1989) An application-oriented guide for designing Lagrangean dual ascent algorithms. *EJOR*, **43**, 197-205.

[49] M.H.Held, P.Wolfe and H.D.Crowder (1974) Validation of subgradient optimization. *Math.Prog.*, **6**, 62-88.

[50] K.Jörnsten and M.Nasberg (1986) A new Lagrangean relaxation approach to the generalized assignment problem. *EJOR*, **27**, 313-323.

[51] H.Reinoso and N.Maculan (1992) Lagrangean decomposition in integer linear programming: a new scheme. *INFOR*, **30**, 1-5.