LAGRANGEAN RELAXATION

J.E. Beasley

The Management School
Imperial College, London

May 1992

1. Introduction

Often in combinatorial optimisation we are dealing with two types of problems:

(a) easy problems, that is problems for which a polynomially bounded algorithm exists; and

(b) hard problems, that is problems for which no polynomially bounded algorithm has yet been found.

By a polynomially bounded algorithm we essentially mean an algorithm for which the number of numeric operations required by the algorithm is a polynomial function of the size of the problem being solved. For example if a problem is of size n then an algorithm requiring n^2 operations is a polynomially bounded algorithm, whilst an algorithm requiring 2^n operations is not a polynomially bounded algorithm.

It has been known since the 1970's that hard problems can be partitioned into two sets:

(a) NP-complete problems; and

(b) others.

The set of NP-complete problems have the property that they are equivalent in the sense that if a polynomially bounded algorithm is ever found for just *one* of the problems in the set then polynomially bounded algorithms exist for *all* the problems in the set.

It is clear that a tremendous amount of research, undoubtedly many person-centuries of effort, has been carried out into NP-complete problems - but no polynomially bounded algorithm for any NP-complete problem has yet been found.

This in turn has led to the conjecture that no such algorithms exist.

For more about this topic, which falls under the general heading of *computational complexity*, see the bibliography.

What then has this to do with lagrangean relaxation, the main subject of this chapter. To see the connection we need to know how we, in general, currently solve NP-complete problems.

Figure 1 illustrates the situation. Suppose that we have some problem instance of a NP-complete problem and further suppose that it is a minimisation problem. If, as in Figure 1, we draw a vertical line representing value (the higher up this line the higher the value) then somewhere on this line is the optimal solution to the problem we are considering.

Exactly where on this line this optimal solution lies we do not know, but it must be somewhere!

Conceptually therefore this optimal solution value divides our value line into two:

(a) above the optimal solution value are upper bounds, values which are above the (unknown) optimal solution value

(b) below the optimal solution value are lower bounds, values which are below the (unknown) optimal solution value.

In order to discover the optimal solution value then any algorithm that we develop must address both these issues i.e. it must concern itself both with upper bounds and with lower bounds.

In particular the quality of these bounds is important to the computational success of any algorithm:

(a) we like upper bounds that are as close as possible to the optimal solution, i.e. as small as possible

(b) we like lower bounds that are as close as possible to the optimal solution, i.e. as large as possible.

1.1 Upper bounds

Techniques for generating upper bounds are essentially beyond the scope of this chapter (but are dealt with elsewhere in this book). Suffice to say here that typically upper bounds are found by searching for feasible solutions to the problem, that is solutions which satisfy the constraints of the problem.

A number of well-known general techniques are available to find feasible solutions to combinatorial optimisation problems, for example:

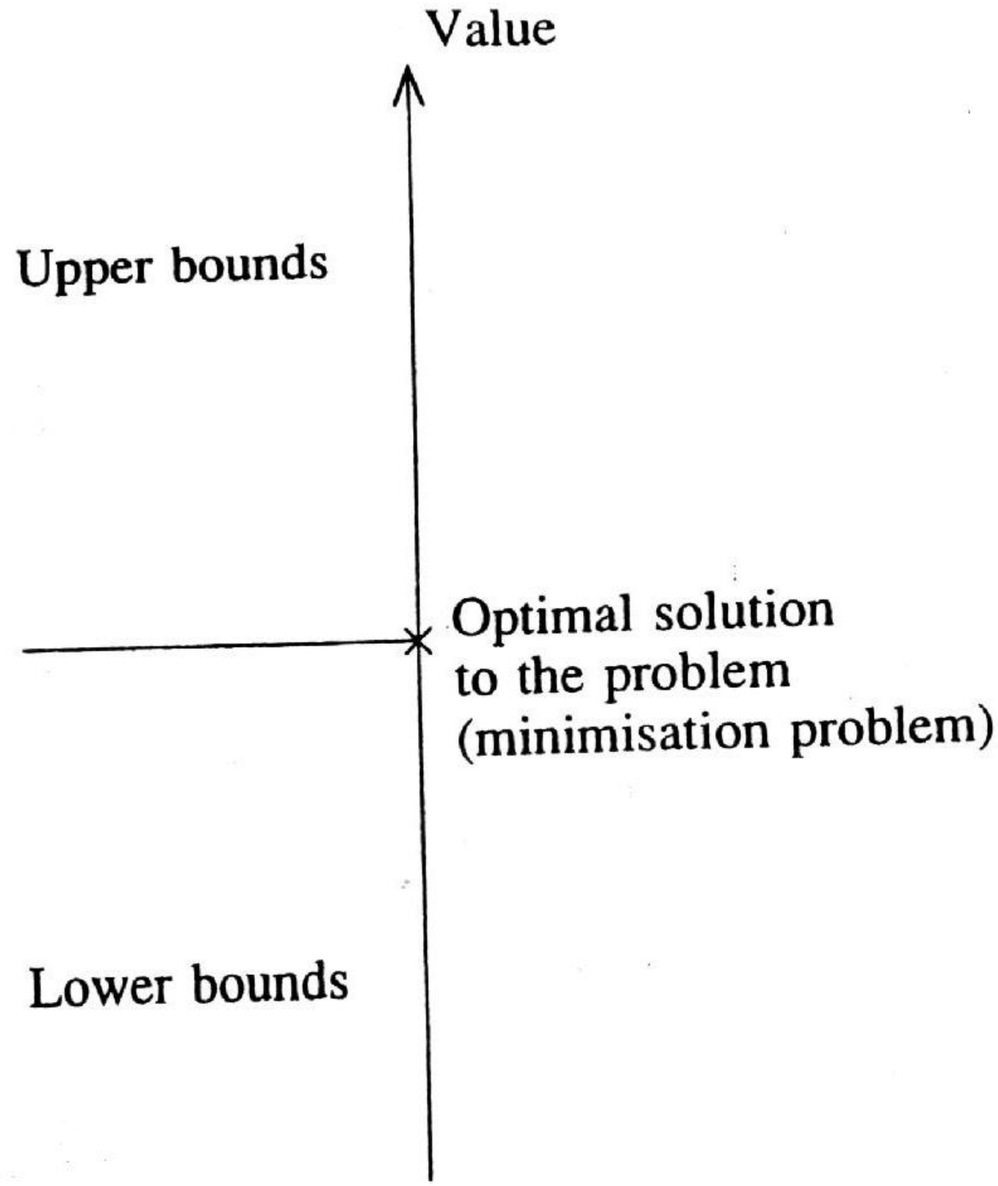

Figure 1

(a) interchange
(b) tabu search
(c) simulated annealing
(d) genetic algorithms.

In addition, for any particular problem, we may well have techniques which are specific to the problem being solved.

1.2 Lower bounds

One well-known general technique which is available to find lower bounds is linear programming relaxation. In linear programming (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 be:
(a) solved exactly using a standard algorithm (simplex or interior point); or
(b) 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 upon at much greater length in this chapter. Suffice to say for the moment that lagrangean relaxation involves:
(a) taking an integer (or mixed-integer) programming formulation of the problem
(b) attaching lagrange multipliers to some of the constraints in this formulation and relaxing these constraints into the objective function
(c) solving (exactly) the resulting integer (or mixed-integer) program.
The solution value obtained from step (c) 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

(a) 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 (c) above).

There are two basic reasons why this approach is well-known (and well-used):
(1) many combinatorial optimisation 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 (b) above) we are left with an easy problem to solve and attention can then be turned to choosing numeric values for the lagrange multipliers.
(2) 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:
(a) subgradient optimisation; and
(b) multiplier adjustment.

Figure 2 summarises the situation. For finding upper bounds (c.f. Figure 1) we have the heuristic techniques mentioned above. For finding lower bounds we have the various relaxations mentioned above.

1.3 Overview

1.3.1 Techniques

In this chapter we deal with:
- lagrangean relaxation
- lagrangean heuristics
- problem reduction
- subgradient optimisation

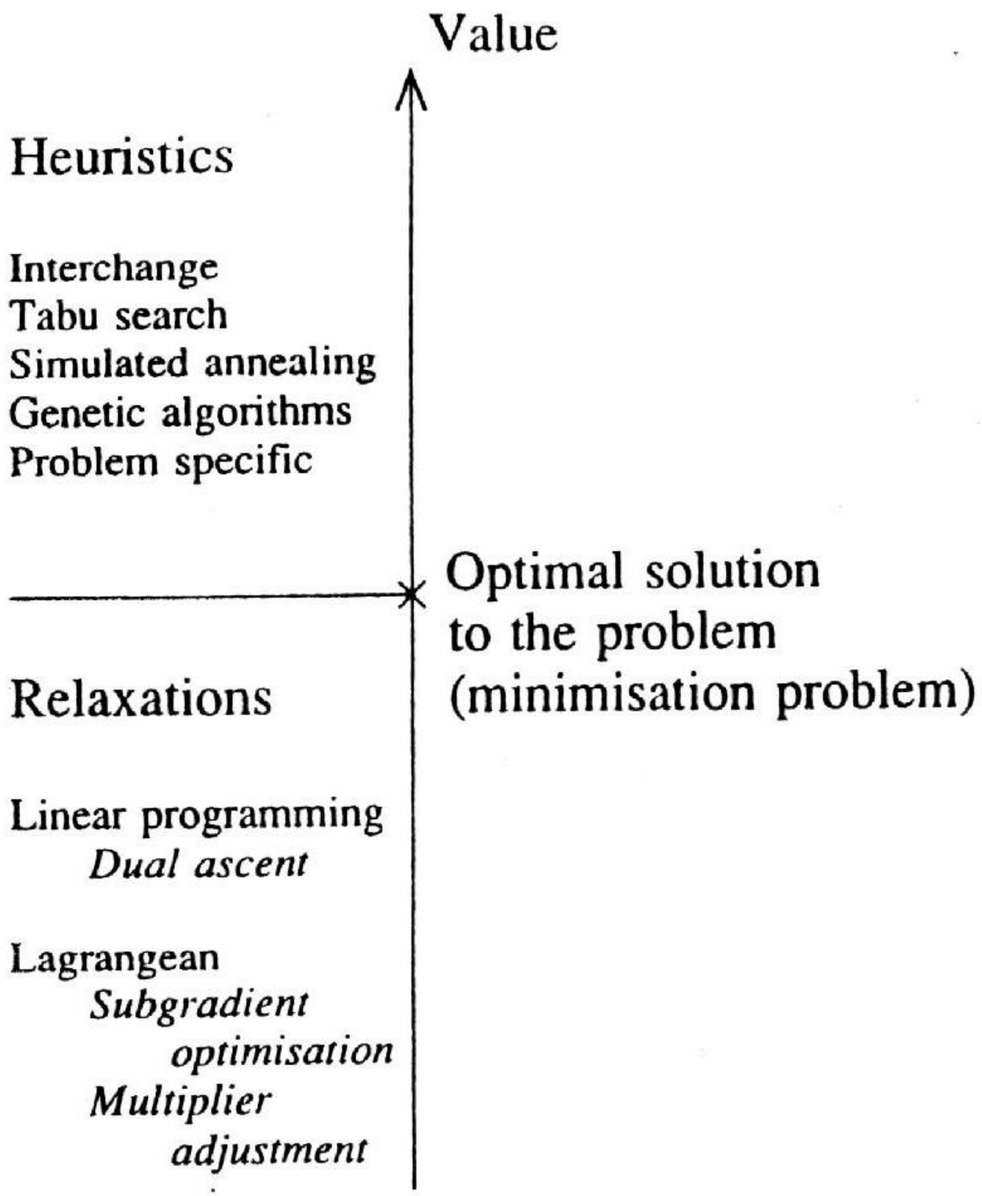

Figure 2

- 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 that takes only minutes to run on a computer.

This issue of *computational effectiveness*, i.e. can an algorithm solve problems in a reasonable time frame, is a key one in algorithm design and development.

We therefore, in this chapter, 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 optimisation problems, namely the set covering problem.

1.3.2 Review

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:

(a) selecting calender year 1991 (the most recent complete calender year at the time of writing);
(b) selecting a number of journals; and
(c) 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.

2. Exposition

2.1 Introduction

Consider the following general zero-one problem (written in matrix notation):

Problem (P) minimise cx
subject to $Ax \geq b$
$Bx \geq d$
$x \in (0,1)$

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:

minimise cx
subject to $Ax \geq b$
$Bx \geq d$
$0 \leq x \leq 1$

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 (problem P).

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

2.2 Lagrangean relaxation

Lagrangean relaxation was developed in the early 1970's with the pioneering work of Held and Karp [41,42] on the travelling salesman problem and is today an indispensable technique for generating lower bounds for use in algorithms to solve combinatorial optimisation 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:

minimise $cx + \lambda(b - Ax)$
subject to $Bx \geq d$
$x \in (0,1)$

i.e. what we have done here is:
(a) to have chosen some set of constraints in the problem for relaxation; and
(b) attached 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 <u>any</u> $\lambda \geq 0$, gives a lower bound on the optimal solution to the original problem P. This can be seen as follows:

The value of

minimise cx
subject to $Ax \geq b$
$Bx \geq d$
$x \in (0,1)$

is greater than the value of

minimise $cx + \lambda(b - Ax)$
subject to $Ax \geq b$
$Bx \geq d$
$x \in (0,1)$

(since as $\lambda \geq 0$ and $(b-Ax) \leq 0$ we are merely adding a term which is ≤ 0 to the objective function)

is greater than the value of

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

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

The program after lagrangean relaxation, namely:

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

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 here 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) 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 $Bx \geq d$.

(b) 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 value of the optimal integer solution. This involves finding multipliers which correspond to:

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

This program is called the lagrangean dual program.

Ideally the optimal value of the lagrangean dual program (a maximisation program) is equal to the optimal value of the original zero-one integer program (a minimisation 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 consider one of the simplest NP-complete combinatorial optimisation problems, namely the set covering problem.

2.3 Set covering problem

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

Defining:

$$\begin{array}{lll} x_j & = 1 & \text{if column j (cost } c_j > 0) \text{ is in the solution} \\ & = 0 & \text{otherwise} \end{array}$$

the SCP is:

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

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

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

$x \in (0,1)$)

(a) to choose some set of constraints in the problem for relaxation; and
(b) to attach lagrange multipliers to these constraints in order to bring them into the objective function.

Step (a) 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,...,m) and so:

(a) we choose this set of constraints for relaxation; and
(b) attach lagrange multipliers $\lambda_i \geq 0$ i=1,...,m to these constraints.

If we do this we find that LLBP is:

$$\text{minimise} \quad \sum_{j=1}^{n} c_j x_j + \sum_{i=1}^{m} \lambda_i (1 - \sum_{j=1}^{n} a_{ij} x_j)$$

$$\text{subject to} \quad x_j \in (0,1) \qquad j=1,\ldots,n$$

i.e.

$$\text{minimise} \quad \sum_{j=1}^{n} [c_j - \sum_{i=1}^{m} \lambda_i a_{ij}] x_j + \sum_{i=1}^{m} \lambda_i$$

$$\text{subject to} \quad x_j \in (0,1) \qquad j=1,\ldots,n$$

$$\text{Defining} \quad C_j = [c_j - \sum_{i=1}^{m} \lambda_i a_{ij}] \qquad j=1,\ldots,n$$

i.e. C_j is the coefficient of x_j in the objective function of LLBP we have that LLBP becomes:

$$\text{minimise} \quad \sum_{j=1}^{n} C_j x_j + \sum_{i=1}^{m} \lambda_i$$

$$\text{subject to} \quad x_j \in (0,1) \qquad j=1,\ldots,n$$

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

In order to generate a lagrangean relaxation of ...

$X_j = 1$ if $C_j \leq 0$
$= 0$ otherwise

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

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

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

Figure 3 summarises 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:

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

$$(a_{ij}) = \begin{vmatrix} 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 1 \end{vmatrix}$$

i.e. the example SCP is:

$$\text{minimise} \quad 2x_1 + 3x_2 + 4x_3 + 5x_4$$
$$\text{subject to} \quad x_1 + x_3 \geq 1$$
$$x_1 + x_4 \geq 1$$
$$x_2 + x_3 + x_4 \geq 1$$
$$x_j \in (0,1) \qquad j=1,\ldots,4$$

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:

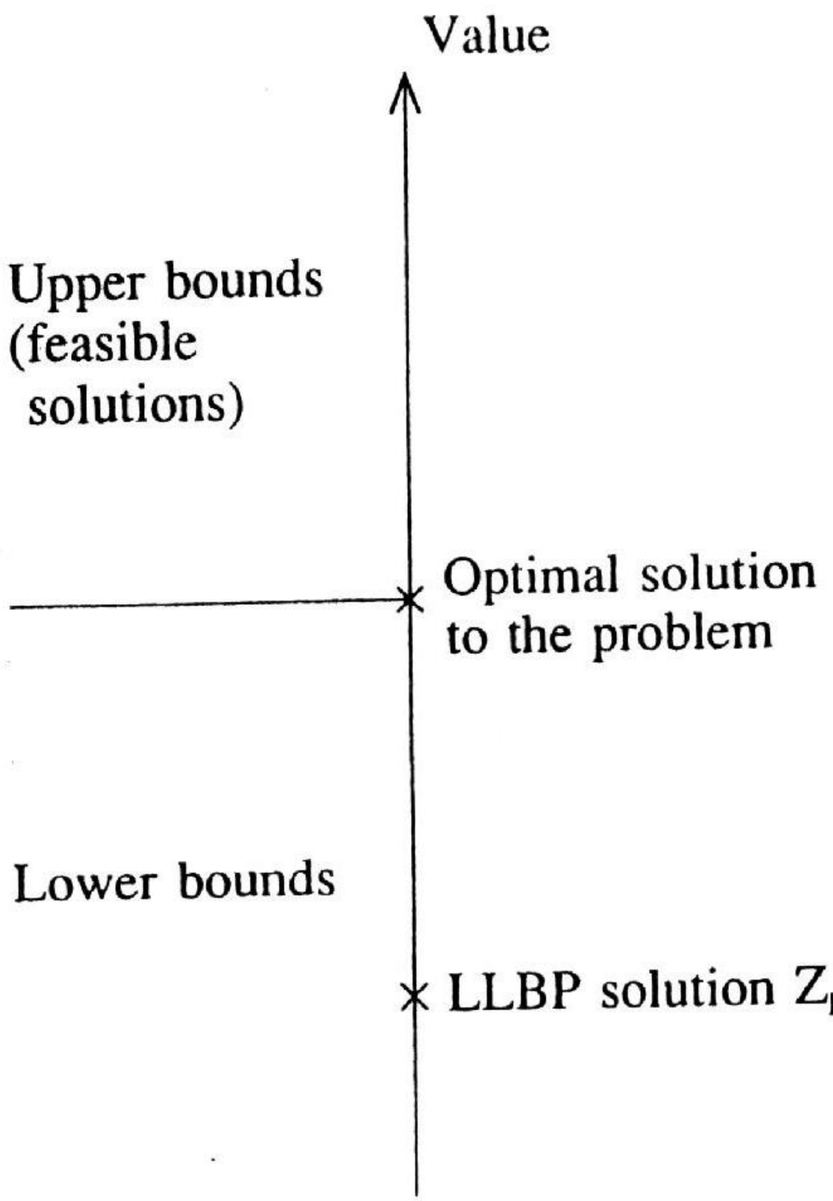

Figure 3

minimise $2x_1 + 3x_2 + 4x_3 + 5x_4$ $+ \lambda_1(1 - x_1 - x_3)$ $+ \lambda_2(1 - x_1 - x_4)$ $+ \lambda_3(1 - x_2 - x_3 - x_4)$

subject to $x_j \in (0,1)$ $j=1,\ldots,4$

i.e. LLBP is

minimise $(2 - \lambda_1 - \lambda_2)x_1 + (3 - \lambda_3)x_2 + (4 - \lambda_1 - \lambda_3)x_3 + (5 - \lambda_2 - \lambda_3)x_4 + \lambda_1 + \lambda_2 + \lambda_3$

subject to $x_j \in (0,1)$ $j=1,\ldots,4$

Hence

$C_1 = (2 - \lambda_1 - \lambda_2)$
$C_2 = (3 - \lambda_3)$
$C_3 = (4 - \lambda_1 - \lambda_3)$
$C_4 = (5 - \lambda_2 - \lambda_3)$

and LLBP is:

minimise $C_1x_1 + C_2x_2 + C_3x_3 + C_4x_4 + \lambda_1 + \lambda_2 + \lambda_3$

subject to $x_j \in (0,1)$ $j=1,\ldots,4$

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

$X_j = 1$ if $C_j \leq 0$
$= 0$ otherwise

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

$$Z_{LB} = C_1X_1 + C_2X_2 + C_3X_3 + C_4X_4 + \lambda_1 + \lambda_2 + \lambda_3$$

2.4 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

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

and

$$Z_{LB} = C_1X_1 + C_2X_2 + C_3X_3 + C_4X_4 + \lambda_1 + \lambda_2 + \lambda_3$$
$$= -1.1 + 0 + 0 + 0 + 1.5 + 1.6 + 2.2$$
$$= 4.2$$

Note here that this value of 4.2 is indeed a lower bound on the optimal solution (which we know is of value 5) to the original SCP.

2.5 Advanced lagrangean relaxation

(1) If we relax equality constraints then λ is unrestricted in sign (i.e. λ 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 incorrect.

For example consider the SCP with 3 rows and 4 columns that we 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:

minimise $cx + \lambda(b - Ax)$
subject to $Bx \geq d$
$x \in (0,1)$

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 $X \in (0,1)$). This feasible solution is of value cX whereas the lower bound obtained from LLBP is of value $[cX + \lambda(b - AX)]$. 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 *only* optimal for the original problem if:
(a) X is feasible for the original problem; and
(b) $cX = [cX + \lambda(b - AX)]$ 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 (a) and (b) 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≤x≤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:

$$\text{minimise} \quad \sum_{j=1}^{n} C_j x_j + \sum_{i=1}^{m} \lambda_i$$

$$\text{subject to} \quad x_j \in (0,1) \qquad j=1,\ldots,n$$

with solution

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

It is clear that replacing $x_j \in (0,1)$ j=1,....,n by $0 \leq x \leq 1$ j=1,....,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 linear programming 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 maximum lower bound = LP relaxation solution
No maximum lower bound ≥ 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:

Problem (P) minimise cx
subject to $Ax \geq b$
$Bx \geq d$
$x \in (0,1)$

then we can set up the table:

there are a number of possible lagrangean relaxations. This is the

Relaxation number	Constraints		Computational effort involved in solving LLBP		Integrality property? (yes/no)
	$Ax \geq b$ Relax? (yes/no)	$Bx \geq d$ Relax? (yes/no)	Number of multipliers	Number of operations	
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 or that set 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 $2^s=2^2=4$ possible relaxations).

Filling in the details for "Computational effort involved in solving LLBP" and "Integrality property? (yes/no)" (the question marks in the above table) depends upon examining each relaxation in mathematical detail. Specifically we need to find:

(a) the number of lagrange multipliers needed - this affects the amount of computer memory needed

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

(c) 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:

(a) relaxation 1 can be ignored since it is just the original problem (no constraints relaxed)

(b) 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

(c) 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

then we can set up the table:

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 (a), (b) and (c) 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 of use 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 above table. To see this consider the original problem:

Problem (P) minimise cx
subject to $Ax \geq b$
$Bx \geq d$
$x \in (0,1)$

and transform this into:

minimise cx
subject to $Ax \geq b$
$y = x$
$By \geq d$
$x \in (0,1)$
$y \in (0,1)$

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

If we now relax (in a lagrangean fashion) the equality constraint linking y and x together (introduce a lagrange multiplier vector λ (unrestricted in sign)) we get the lagrangean lower bound program

(LLBP):

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

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

$$\begin{array}{lll} \text{minimise} & (c + \lambda)x & + \\ \text{subject to} & Ax \geq b & \\ & x \in (0,1) & \end{array} \qquad \begin{array}{lll} \text{minimise} & -\lambda y & \\ \text{subject to} & By \geq d & \\ & y \in (0,1) & \end{array}$$

and 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:

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

(b) 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})

(c) considering relaxing 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

(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:

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

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

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

is valid (but is redundant in the original formulation). Whereas LLBP for the SCP was given before by:

$$\begin{array}{lll} \text{minimise} & \sum_{j=1}^{n} C_j x_j + \sum_{i=1}^{m} \lambda_i & \\ \text{subject to} & x_j \in (0,1) & j=1,\ldots,n \end{array}$$

adding this constraint to LLBP gives:

$$\begin{array}{lll} \text{minimise} & \sum_{j=1}^{n} C_j x_j + \sum_{i=1}^{m} \lambda_i & \\ \text{subject to} & 1 \leq \sum_{j=1}^{n} x_j \leq m & \\ & x_j \in (0,1) & j=1,\ldots,n \end{array}$$

It is clear that:

s(s-1)/2 equality linking constraints and hence 2 possible lagrangean decompositions.

(a) 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

(b) this new (strengthened) LLBP is easy to solve computationally (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).

2.6 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.

2.6.1 Lagrangean heuristic

In a lagrangean heuristic we take the solution to LLBP and attempt to convert (transform) 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 (c.f. Figure 3).

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 consider 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:

(a) set $S=[\ j \mid X_j=1\ j=1,\ldots,n]$

It is clear that:

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

(c) 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 of $X_1=1$, $X_2=X_3=X_4=0$ we get:

(a) $S=[1]$

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

(c) $S=[1,2]$ is now a feasible solution to the original SCP of cost $c_1 + c_2 = 2 + 3 = 5$

Fortuitously here we have, via our lagrangean heuristic, 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 [10-15,19,20]) has been that relatively simple lagrangean heuristics can give good quality results.

2.6.2 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:

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

(b) fixing x_k to zero for some k enables us to delete column k from the problem.

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

The basic idea here is to make any reductions such that combining the optimal solution to the reduced problem with:
(a) $x_k=1 \ \forall k \in R_1$; and
(b) $x_k=0 \ \forall k \in R_0$
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 optimisation we are often dealing with large problems, that is problems with an enormous number of potential solutions. Our task is to find the optimal solution.

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 you 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. The reader can tax their own calculator in finding a value for 2^{3000}!

Informally, one can think of looking for the optimal solution to a combinatorial optimisation 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. Hopefully this will mean that we will find the needle quicker!

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.

Problem reduction, involving fixing x_k to zero, 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 $Z_{LB}+C_k$ if $X_k=0$. (Hint: solve LLBP with the additional constraint that $x_k=1$).

(2) Then 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

problem.

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 4 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 i.e. column k cannot be part of the optimal solution, so it can be removed from the problem and the optimal solution to the reduced (SCP) problem will be the same as the optimal solution to the original (SCP) 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 $R_1=\emptyset$.

For this solution we had C_1=-1.1, C_2=0.8, C_3=0.3 and C_4=1.2 (see above).

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

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

Problem reduction, involving fixing x_k to one, can be carried out as follows:

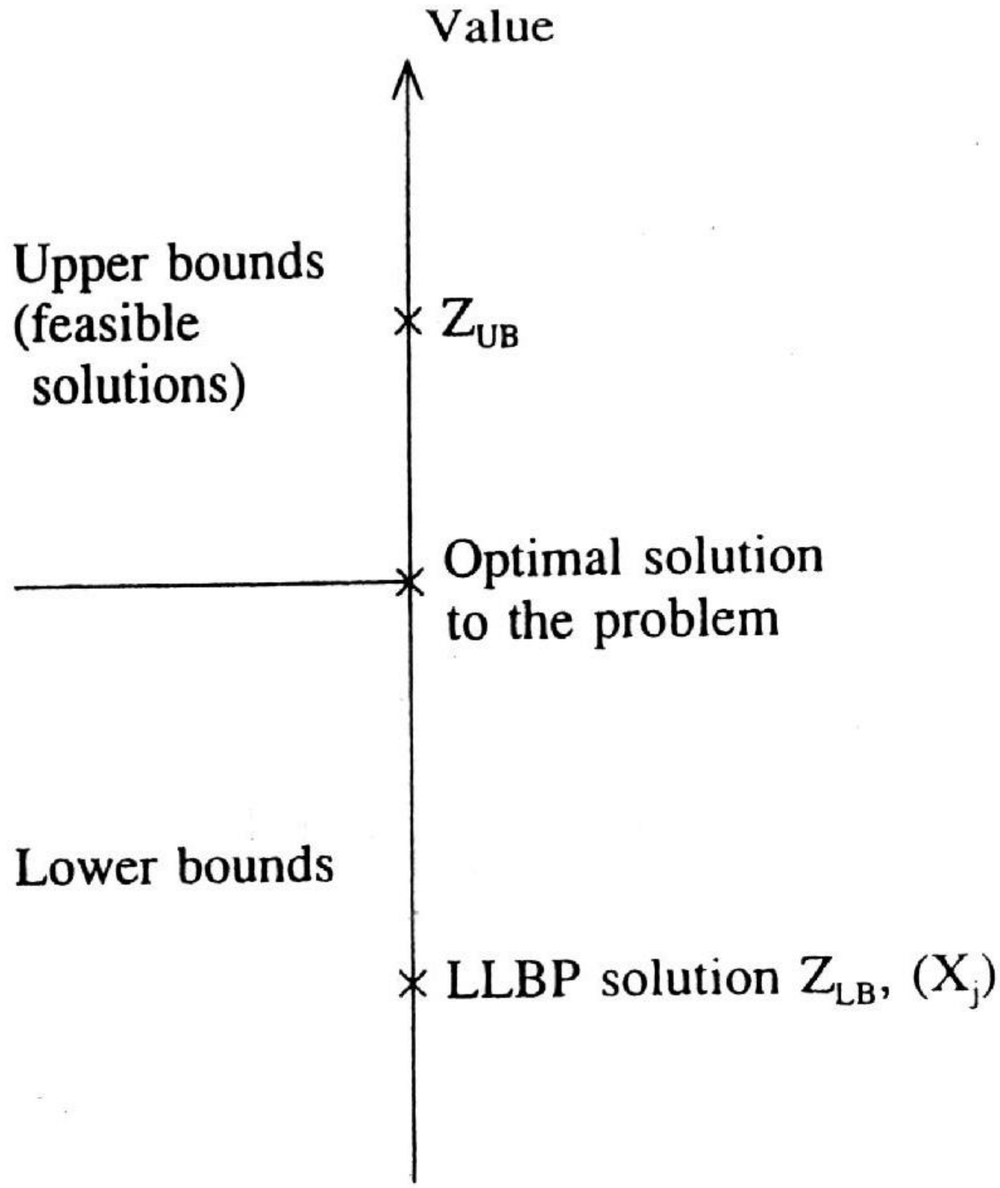

Figure 4

(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 to one) 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 that the above is correct follows exactly the same lines as before (for the reduction involving fixing x_k to zero) and will be left to the reader.

To illustrate problem reduction involving fixing x_k to zero consider our example LLBP solution with X_1=1, X_2=X_3=X_4=0, Z_{LB}=4.2, C_1=-1.1, C_2=0.8, C_3=0.3, C_4=1.2 and Z_{UB}=5 (see above). Hence:

Z_{LB}-C_k = 4.2 - (-1.1) = 5.3 for k=1 (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∪[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 [10-15,19,20]) has been that problem reduction significantly improves the computational performance of any algorithm. For example the algorithm given in [14] for the SCP includes problem reduction and typically over 75% (often well over 75%) of the variables are eliminated by problem reduction.

2.6.3 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, in percentage terms, 16% (100(5-4.2)/5) away from the optimal solution of 5 we were able to gain important information, namely:
(a) actually finding the optimal solution via the lagrangean heuristic; and
(b) making 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 as lucky the point is clear - useful information can be deduced from LLBP solutions which, in lower bound terms, are quite far from the optimal solution.

lagrangean lower bound program solution can be used to yield useful

3. Deciding lagrange multipliers

3.1 Introduction

In the previous section we have seen how to apply lagrangean relaxation to:
(a) generate a lower bound;
(b) generate a upper bound (corresponding to a feasible solution); and
(c) 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 can we find numerical values for the multipliers.

There are two basic approaches to deciding values for the lagrange multipliers (λ_i):
(a) subgradient optimisation; and
(b) multiplier adjustment.
We deal with each in turn.

3.2 Subgradient optimisation

Recall the original problem that we are attempting to solve:

minimise cx
subject to $Ax \geq b$
$Bx \geq d$
$x \in (0,1)$

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

minimise $cx + \lambda(b - Ax)$
subject to $Bx \geq d$
$x \in (0,1)$

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

Subgradient optimisation is an iterative procedure which, from a initial set of multipliers, involves generating further lagrange multipliers in a systematic fashion. It can be viewed as a procedure which attempts to maximise 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 $\sum_{j=1}^{n} a_{ij}x_j \geq b_i$ $(i=1,\ldots,m)$, the basic subgradient optimisation iterative procedure is as follows:

(1) Let π be a user decided parameter satisfying $0 < \pi \leq 2$. Initialise 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 (λ_i) of multipliers, 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:

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

(4) Define a (scalar) step size T by

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

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 the $\sum_{i=1}^{m} (G_i)^2$ factor

being a scaling factor.

(5) Update λ_i using

$\lambda_i = \max(0, \lambda_i + TG_i)$ i=1,...,m

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

As currently set out the above iterative procedure would never terminate. In fact we introduce a termination rule based upon either:
(a) limiting the number of iterations that can be done; or
(b) the value of π (reducing π during the course of the procedure and terminating when π is small, see below).

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

(1) Let π=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 λ_1=1.5, λ_2=1.6, λ_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:

$G_1 = (1 - X_1 - X_3) = 1 - 1 - 0 = 0$
$G_2 = (1 - X_1 - X_4) = 1 - 1 - 0 = 0$
$G_3 = (1 - X_2 - X_3 - X_4) = 1 - 0 - 0 - 0 = 1$

(4) The step size T is given by:

$T = 2(6 - 4.2)/(0^2 + 0^2 + 1^2) = 3.6$

(5) Updating λ_i using $\lambda_i = \max(0, \lambda_i + TG_i)$ gives:

$\lambda_2 = \max(0, 1.6 + 3.6(0)) = 1.6$
$\lambda_3 = \max(0, 2.2 + 3.6(1)) = 5.8$

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 optimisation i.e. 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 go negative.

However, suppose that we let Z_{max} be the maximum lower bound found over all subgradient iterations (where initially Z_{max}=-∞ and we update Z_{max} at each subgradient iteration using Z_{max}=max(Z_{max}, Z_{LB})).

What 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.

However it is common for Z_{max} to approach very close to (or even attain) the maximum lower bound possible from the lagrangean lower bound program, i.e. for Z_{max} to approach very close to (or even attain) the value of the lagrangean dual program.

Figure 5 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.

3.3 Advanced subgradient optimisation

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

(1) It is clear that the subgradient optimisation procedure can be

$\lambda_1 = \max(0,\ 1.5 + 3.6(0)) = 1.5$

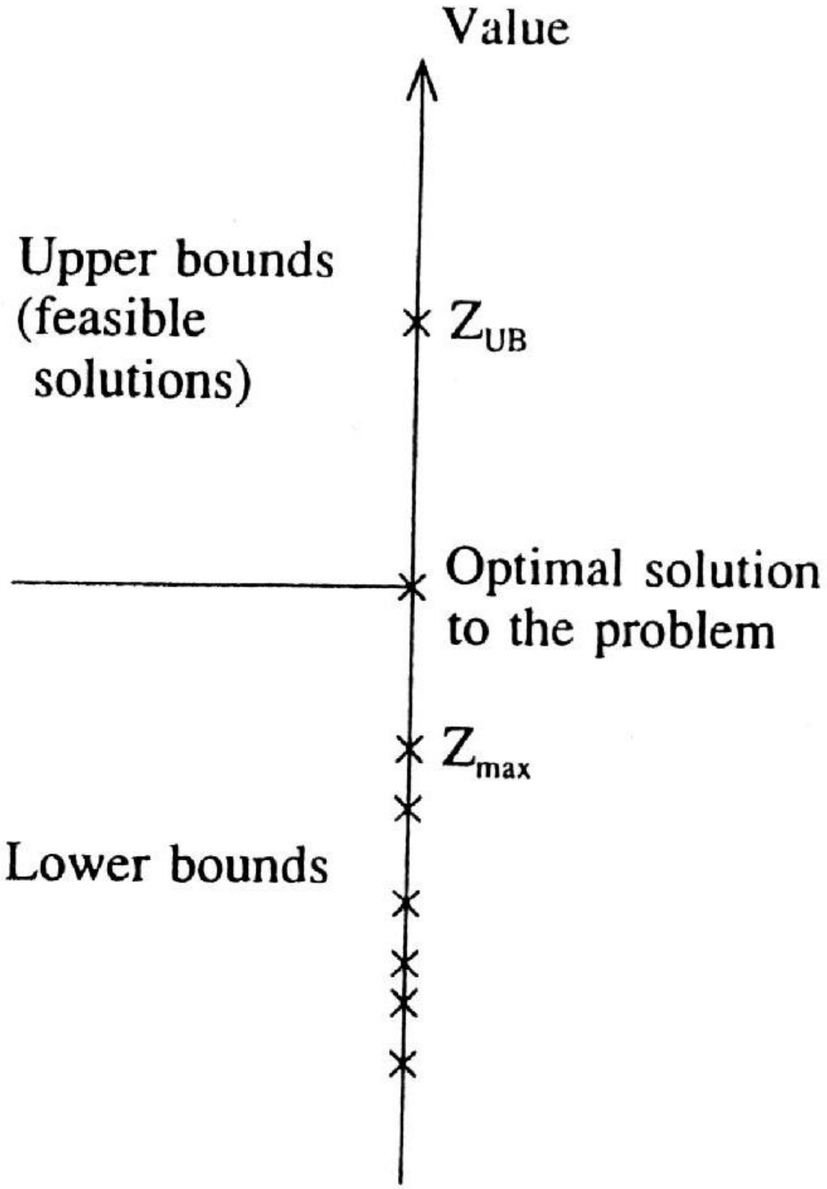

Figure 5

(1) It is clear that the subgradient optimisation procedure can be

terminated if we find that $Z_{max} = Z_{UB}$. In this case we have that 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. This can be seen from Figure 5, 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 π=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 you are considering.

(3) Unless terminated by (1) above terminate the subgradient optimisation procedure when π is small (e.g. $\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) My experience is that how the lagrange multipliers are initialised is not particularly important, i.e. how quickly the subgradient procedure terminates and the quality of the final lower bound (Z_{max}) obtained is relatively insensitive to the initial choice of multipliers.

(5) It is helpful to replace Z_{UB} in the equation $T = \pi(Z_{UB} - Z_{LB})/\sum_{i=1}^{n} (G_i)^2$ 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} if the equation for the step size is as given above [$T = \pi(Z_{UB} - Z_{LB})/ \sum_{i=1}^{m} (G_i)^2$] then T gets smaller and smaller, implying 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 ($\lambda_i = \max(0, \lambda_i + TG_i)$). 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:

$G_i=0$ if $\lambda_i=0$ and $G_i<0$

(7) It may happen that, at some subgradient iteration, $G_i=0$ i=1,...,m (assuming that the adjustment referred to above has been carried out). If so then the subgradient procedure terminates. If this happens then:
   (a) 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)
   (b) otherwise we simply have a lower bound Z_{max} upon 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 <u>scaled</u> so that their right-hand sizes have a maximum absolute value of one (or are zero).

This is because if you are relaxing constraints with varying right-hand sides a constraint with a very large right-hand side can dominate the step size expression for T.

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

$$\lambda_i = \lambda_i + TG_i \qquad i=1,...,m$$

(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 before 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:
   (a) found from the lagrangean heuristic a feasible solution of value 5 (this 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 $Z_{UB}=5$)
   (b) 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 optimisation has been much used in the literature in conjunction with lagrangean relaxation. The reasons for this are essentially twofold:
   (a) it works, i.e. subgradient optimisation in conjunction with lagrangean relaxation give good quality lower bounds for a wide variety of combinatorial optimisation problems
   (b) subgradient optimisation works from a general description

[$\sum_{j=1}^{n} a_{ij}x_j \geq b_i$ (i=1,...,m)] of the relaxed constraints and as

dominate the step size expression for T.

such is capable of being directly applied, without alteration, whatever the exact nature of those constraints (c.f. multiplier adjustment below).

3.3 Multiplier adjustment

Multiplier adjustment is simply a heuristic that:
(a) given a starting set of lagrange multipliers;
(b) attempts to improve them in some systematic way so as to generate an improved lower bound; and
(c) if an improvement is made repeats (b) above.

Often we simply change a single multiplier at each iteration, c.f. subgradient optimisation where we (potentially) change all multipliers at each iteration.

The advantages of multiplier adjustment are:
(a) usually computationally cheap; and
(b) usually get an increase (or at least no decrease) in the lower bound at each iteration.

The price we pay for this advantage is:
(a) the final lower bound obtained can be poor (i.e. worse than that obtained from subgradient optimisation); and
(b) different problems require different multiplier adjustment algorithms (unlike subgradient optimisation 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.

To illustrate multiplier adjustment we shall develop a multiplier adjustment algorithm for the SCP.

As in developing lagrangean heuristics developing multiplier adjustment algorithms is an art. However, exactly as for subgradient optimisation above, where the equation for updating multipliers was:

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

the direction in which we would like to change multipliers is clear:
(i) if $G_i<0$ we would like to reduce λ_i
(ii) if $G_i=0$ we leave λ_i unchanged
(iii) if $G_i>0$ we would like to increase λ_i
c.f. the above subgradient optimisation equation for multiplier update.

Hence one possible (very simple) multiplier adjustment algorithm for the SCP is:
(a) solve LLBP with the current set of multipliers (λ_i)
(b) choose any row i for which $G_i>0$ (i.e. row i is uncovered in the current LLBP solution)
(c) if row i is uncovered then it is easy to see from the relevant mathematics of LLBP that:
  (1) increasing λ_i will increase the lower bound obtained from LLBP; and
  (2) the maximum amount (δ) by which we can increase λ_i before the solution to LLBP changes is given by:
    $\delta=\min(C_j \mid a_{ij}=1\ j=1,\ldots,n)$
    i.e. $\delta=\min(C_j \mid$ column j covers row i)
(d) increase λ_i by δ and go to (a).

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 of $\lambda_1=1.5$, $\lambda_2=1.6$ and $\lambda_3=2.2$ that we had before:

(a) 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 $G_1=0$, $G_2=0$, $G_3=1$
(b) row 3 is uncovered as $G_3>0$
(c) columns 2, 3 and 4 cover row 3 so $\delta=\min(C_2, C_3, C_4) = \min(0.8, 0.3, 1.2) = 0.3$
(d) so we increase λ_3 by 0.3 to give a new set of multipliers of $\lambda_1=1.5$, $\lambda_2=1.6$, $\lambda_3=2.5$

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.

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 investigating 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.

Note here that, as remarked above, unlike subgradient optimisation where we simply apply a sequence of general formulae for subgradients, step size and lagrange multiplier update, we have that multiplier adjustment algorithm design is a much more creative (difficult!) process.

4. Dual ascent

4.1 Introduction

Dual ascent came to prominence with the work of Erlenkotter [26] (and Bilde and Krarup [16]) on the uncapacitated warehouse location problem which, computationally, was very successful.

Consider the linear programming (LP) relaxation of any combinatorial optimisation problem P (which is a minimisation problem). As P is a minimisation problem the LP relaxation is also a minimisation problem. The dual linear program associated with the LP relaxation is therefore a maximisation problem. Hence:

optimal P (integer) solution
≥
LP relaxation solution
=
dual LP solution
≥
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 illustrates the situation. In that figure we essentially have three regions:

(a) upper bounds, the region above the optimal (integer) solution;
(b) dual LP feasible solutions, the region below the LP relaxation solution; and
(c) 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.

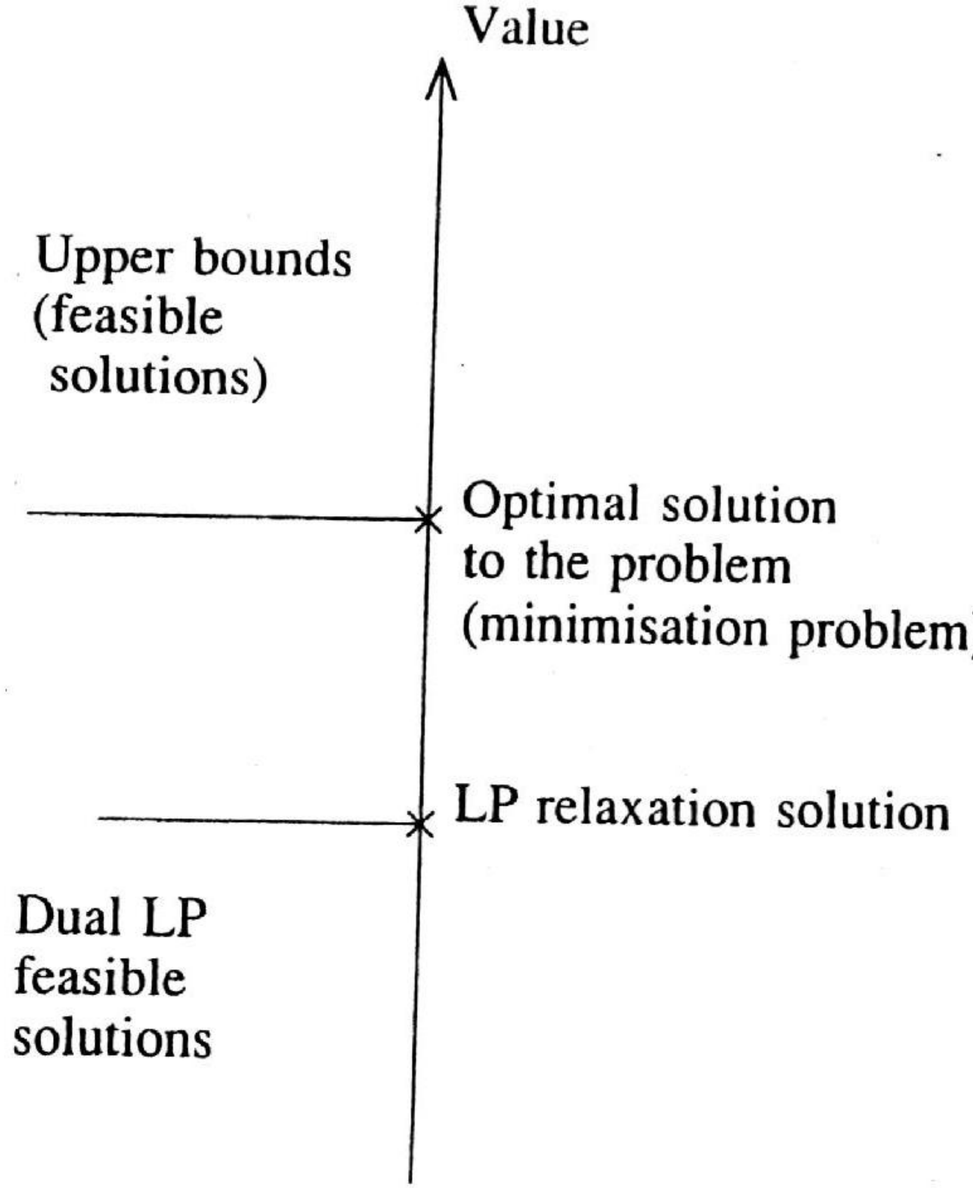

Figure 6

problem.

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

$$\text{minimise} \quad \sum_{j=1}^{n} c_j x_j$$

$$\text{subject to} \quad \sum_{j=1}^{n} a_{ij} x_j \geq 1 \qquad i=1,\ldots,m$$

$$0 \leq x_j \leq 1 \qquad j=1,\ldots,n$$

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

$$\text{minimise} \quad \sum_{j=1}^{n} c_j x_j$$

$$\text{subject to} \quad \sum_{j=1}^{n} a_{ij} x_j \geq 1 \qquad i=1,\ldots,m$$

$$0 \leq x_j \qquad j=1,\ldots,n$$

and the dual LP is:

$$\text{maximise} \quad \sum_{i=1}^{m} u_i$$

$$\text{subject to} \quad \sum_{i=1}^{m} u_i a_{ij} \leq c_j \qquad j=1,\ldots,n$$

$$u_i \geq 0 \qquad i=1,\ldots,m$$

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

4.2 Example dual ascent algorithm

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

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

(b) take each u_i (i=1,...,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 our example SCP the dual LP is:

maximise $u_1 + u_2 + u_3$
subject to $u_1 + u_2 \leq 2$
$u_3 \leq 3$
$u_1 + u_3 \leq 4$
$u_2 + u_3 \leq 5$
$u_1, u_2, u_3 \geq 0$

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

(a) set $u_1=u_2=u_3=0$
(b) (1) the constraints involving u_1 are:
$u_1 \leq 2$
$u_1 \leq 4$
(after setting $u_2=u_3=0$) so that u_1 can be increased to 2
(2) the constraints involving u_2 are:
$u_2 \leq 0$
$u_2 \leq 5$
(after setting $u_1=2$ and $u_3=0$) so u_2 cannot be increased
(3) the constraints involving u_3 are:
$u_3 \leq 3$
$u_3 \leq 2$
$u_3 \leq 5$
(after setting $u_1=2$ and $u_2=0$) 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 [illegible] not just increasing u_i as above, but also

investigating reducing u_i (thereby enabling us to increase other u_j's). Discovering whether a particular dual ascent algorithm gives good quality lower bounds at reasonable computational cost is a matter for computational experimentation.

4.3 Connections

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

(a) lagrangean relaxation (with the multipliers being decided by subgradient optimisation or multiplier adjustment); and
(b) dual ascent, i.e. heuristically solve 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:

(a) 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:

(b) the values of the lagrange multipliers that maximise 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 <u>if</u> the lagrangean relaxation has the integrality property <u>then</u> 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.

Plainly we could have designed a better dual ascent algorithm, for example investigating not just increasing u_i as above, but also

4.4 Subgradient optimisation or multiplier adjustment or dual ascent?

Which of these three techniques should we use for obtaining lower bounds?

I have to confess that my personal experience has been that subgradient optimisation has always appeared to give me 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 I have never been very keen on multiplier adjustment methods although they appear useful for some problems, e.g. the generalised assignment problem [29,40].

The only time I have tried dual ascent (for the p-median problem) it was a miserable failure!

There is a deeper point here. Some techniques (such as subgradient optimisation, 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, instead only being successful (possibly outstandingly successful) on one or two problems.

Based on this point then, if you have to choose between these three lower bound techniques, my advice would be:

Subgradient optimisation will nearly always work

Multiplier adjustment may work

Dual ascent will probably not work

This advice is supported by evidence from the literature. If you look at the "best" (most successful) optimal algorithm for a number of different

problems you will find that it is usually based on subgradient optimisation and only occasionally based on multiplier adjustment or dual ascent.

4.5 Conclusions

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

4.5.1 Heuristic

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:

(a) Z_{UB} is the value of the best feasible solution found;
(b) Z_{max} is the value of the best lower bound (on the optimal solution to the original problem) found; and
(c) the gap between Z_{UB} and Z_{max} (e.g. $100(Z_{UB}-Z_{max})/Z_{max}$ (in percentage terms)) gives an indication of the quality of the lagrangean heuristic solution (Z_{UB}) obtained.

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 below).

However we can go further than simply accepting whatever quality guarantee figure we end up with after the subgradient procedure (remember that we have no control/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:

(a) suppose we are prepared to accept a solution within α% of optimal, i.e. we would accept a solution Z_{UB} with (Z_{UB}-optimal) $\leq$

(α/100)optimal

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

(c) if this is done then the subgradient procedure will terminate with:

(1) Z_{UB} proved to be within α% of optimal, i.e. we will have achieved our desired (pre-specified) quality guarantee; or

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

4.5.2 Optimal

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

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

(b) if not then we need to resort to a tree search procedure to resolve the problem.

We consider tree search procedures below.

5. Tree search

5.1 Introduction

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 optimisation. 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.

(1) *Tree structure*

I always use a binary tree search. I have found that this "works for me" in all the cases I have tried.

(2) *Branching node*

I always use a depth-first strategy (that is branch from the last available node in the tree). Again I have found that this "works for me" in all the cases I have tried.

I would only recommend using a more complex tree search if you are in despair at the results from a binary depth-first tree search.

(3) *Backtracking*

We can backtrack from a tree node if:

(a) the tree node is infeasible; or

(b) the lower bound at the tree node exceeds the upper bound.

Note here that the solution to the lagrangean lower bound program at a particular tree node being a feasible solution to the original problem is *not* a condition for backtracking (see the fallacy mentioned above under

advanced lagrangean relaxation).

(4) *Forward branching rule*

I always use 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 then things are more complicated than this. Recall the lagrangean lower bound program (LLBP):

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

In a tree search procedure based upon a lagrangean relaxation bound a reasonable rule to use is:

(a) 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.

(b) Consider the vector of values $\lambda(b-AX)$ (note that this is equal to λG) 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.

(c) 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 $(c-\lambda A)$ out of those variables in S (i.e. is most "likely" to be one in an optimal completion of the current tree node).

Note here that, 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).

If you are programming a tree search procedure based upon lagrangean relaxation then my 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 with $\lambda_1=1.5$, $\lambda_2=1.6$, $\lambda_3=2.2$, $X_1=1$, $X_2=X_3=X_4=0$ and $Z_{LB}=4.2$. What variable would we choose to branch on?

(a) We have $\lambda_1=1.5$, $\lambda_2=1.6$, $\lambda_3=2.2$ and $X_1=1$, $X_2=X_3=X_4=0$. In fact we also know that $C_1=-1.1$, $C_2=0.8$, $C_3=0.3$ and $C_4=1.2$ (see above).

(b) Then:

$$\begin{aligned} \lambda_1(1 - X_1 - X_3) &= 1.5(1 - 1 - 0) = 0 \\ \lambda_2(1 - X_1 - X_4) &= 1.6(1 - 1 - 0) = 0 \\ \lambda_3(1 - X_2 - X_3 - X_4) &= 2.2(1 - 0 - 0 - 0) = 2.2 \end{aligned}$$

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).

(c) S=[j | x_j appears in constraint 3, X_j=1]

gives S=∅ since only columns (variables) 2, 3 and 4 appear in constraint 3 and they are all currently zero (X_2=X_3=X_4=0). Hence:

S=[j | x_j appears in constraint 3, X_j=0]

gives S=[2,3,4] and these variables have LLBP objective function coefficients of C_2=0.8, C_3=0.3 and C_4=1.2. The minimum of these values is C_3=0.3, therefore we would choose x_3 to set to one in the forward branch.

(5) *Lower bound computation*

(a) *Initial tree node*

At the initial tree node (before branching) my experience is that it is well worthwhile 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.

(b) *Other tree nodes*

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 I tend to use the strategy of:

(a) initialising the lagrange multipliers at each tree node with the set associated with the best (maximum) lower bound found at the predecessor tree node;

(b) 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;

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

The logic for (c) 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 (c.f. 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.

5.2 Computer programs

Clearly a number of readers of this chapter/book are likely to be doctoral students who are not only:

(a) gaining additional knowledge; but

(b) who may also decide to apply a number of the techniques given in this chapter to their own particular research problem.

For this reason we are 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:

One option is to simply repeat, at each tree node, *exactly* the same

(a) solving the lagrangean relaxation
(b) problem reduction
(c) tree search.

We deal with each in turn.

5.2.1 Solving the 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 lagrangean lower bound program 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 software yourself (software development being both time-consuming and frustrating!).

A excellent source of information on available commercial software is the advertisements/software reviews in "OR/MS Today" (the magazine associated with membership of the American Operations Research Society).

Table 1 gives brief details of additional possible useful software.

5.2.2 Problem reduction

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

(a) 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 LLBP (nor, presumably, in any solution found by the lagrangean heuristic).

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

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 you have complete access to the FORTRAN source code (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.

Generalised assignment problem
A code is available from Guignard and Rosenwein (Operations Research 37 (1989) 658-663).

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 pc's, are readily available, e.g. see the review by Beasley in the Journal of the Operational Research Society 39(5) (1988) 487-509.

There are a number of 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.

Table 1: Useful software

(b) if a variable x_k is fixed to one then:
  (1) any relaxed constraint i which is guaranteed to be always satisfied because $x_k=1$ can be:
    (i) neglected in LLBP simply by always setting $\lambda_i=0$
    (ii) 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:
    (i) 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 <u>always</u> automatically appear in the solution to LLBP (for example for the SCP given above $v_k=0$ would be appropriate)
    (ii) add C_{add} minus the sum of all the v_k's to the solution value given by LLBP to get a valid lower bound
    (iii) ensure that all x_k's 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. to explicitly 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:
(a) the computational effort involved in altering the data structure; against
(b) 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 [illegible] 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. try different strategies and see which is best. However you should be aware that such a question needs to be addressed.

5.2.3 <u>Tree search</u>

A FORTRAN code for a binary depth-first tree search procedure is given in the Appendix.

benefits (computational reduction) from the procedure is [illegible]
consideration in effective algorithm design.

6. Applications

6.1 Introduction

As mentioned at the very beginning of this chapter 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, in this section, have adopted the strategy of:

(a) selecting calender year 1991 (the most recent complete calender year at the time of writing);
(b) selecting a number of journals; and
(c) 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:

(a) we believe that lagrangean relaxation has wide applicability; and
(b) a systematic survey (rather than "selected highlights") provides *evidence* for such a belief.

By the end of this chapter readers can judge for themselves whether lagrangean relaxation has wide applicability or not.

6.2 Journals

The journals that we have selected to systematically review are:

(a) Operations Research
(b) Management Science
(c) European Journal of Operational Research

These journals are some of the premier/most cited journals in the field of operations research/management science.

6.2.1 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.

(1) Ahmadi and Matsuo [2] 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 optimisation and a lagrangean heuristic.

Computational results are presented for problems based on real-world data for large-scale circuit board manufacturing.

(2) Balas and Saltzman [6] present a paper dealing with the three-index assignment problem. This is a generalisation 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 optimisation, dual ascent and a lagrangean heuristic in conjunction with a tree search procedure.

Computational results are given for randomly generated problems.

(3) Shulman [55] 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 optimisation and lagrangean heuristics.

Computational results are presented for a number of randomly generated problems.

(4) Altinkemer and Gavish [4] 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 heuristic algorithms for the problem. A lower bound, based upon lagrangean relaxation and subgradient optimisation, 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).

(5) Noon and Bean [50] present a paper dealing with the asymmetric generalised travelling salesman problem. This is a generalisation of the well-known travelling salesman problem to the problem of deciding a minimum cost travelling salesman tour around sets of cities, where 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 optimisation, problem reduction and a lagrangean heuristic in conjunction with a tree search procedure.

Computational results are presented for a number of randomly generated problems.

(6) Ahmadi and Tang [3] present a paper dealing with the operation

partitioning problem. This is the problem of assigning operations to machines so as to minimise the total movement of jobs between machines.

They present two zero-one formulations of the problem and use lagrangean relaxation, subgradient optimisation and lagrangean heuristics.

Computational results are presented for a number of randomly generated problems.

6.2.2 Management Science

In 1991 Management Science published 115 papers (of assorted lengths) 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.

(1) Pirkul and Schilling [51] 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 present a zero-one formulation of the problem and use lagrangean relaxation, subgradient optimisation 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.

(2) Campbell and Mabert [17] 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 optimisation and a lagrangean heuristic.

Computational results are presented for a number of problems, including some based on data from the Ford Motor company.

(3) Gavish and Pirkul [33] present a paper dealing with the multi-resource generalised 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 use lagrangean relaxation, subgradient optimisation, lagrangean heuristics and problem reduction in conjunction with a tree search procedure.

Computational results are presented for a number of randomly generated problems.

(4) Domich, Hoffman, Jackson and McClain [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 present a zero-one formulation of the problem and use lagrangean relaxation, subgradient optimisation and a lagrangean heuristic.

Computational results are presented for real-world data based upon locating IRS offices in Florida.

(5) Bard and Bejjani [8] present a paper dealing with the problem of leasing telecommunication lines as to achieve a desired level of service at minimum cost.

They a present dynamic programming formulation of the problem which involves solving a number of general integer programs. These general integer programs are solved using lagrangean relaxation, subgradient optimisation and a lagrangean heuristic.

Computational results are presented for real-world data involving a telecommunications company based in Texas.

(6) Drexl [25] 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).

6.2.3 European Journal of Operational Research

In 1991 the European Journal of Operational Research published 207 papers (of assorted lengths) 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.

(1) Cornuejols, Sridharan and Thizy [21] 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 optimisation and a lagrangean heuristic.

Computational results are presented for a number of problems (both taken from the literature and randomly generated).

(2) Aboudi, Hallefjord and Jörnsten [1] 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).

(3) Sharma [54] 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 optimisation and a lagrangean heuristic.

Computational results are presented for problems based upon real-world data.

(4) Deckro, Winkofsky, Hebert and Gagnon [23] present a paper dealing with the multi-project resource constrained scheduling problem. This is the problem of minimising 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.

(5) Lozano, Larraneta and Onieva [48] 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).

(6) Wright and von Lanzenauer [56] 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.

(7) Current and Pirkul [22] 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 transshipment facilities must be located on the primary path.

They present a zero-one formulation of the problem and use lagrangean relaxation, subgradient optimisation and lagrangean heuristics.

Computational results are presented for a number of randomly generated problems.

(8) Barcelo, Fernandez and Jörnsten [7] 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 optimisation, a lagrangean heuristic and problem reduction in conjunction with a tree search procedure.

Computational results are presented for a number of randomly generated problems.

(9) Millar and Gunn [49] 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 optimisation.

Computational results are presented for real-world data involving a large fish-processing company based in Canada.

6.3 Conclusions

In order to summarise the results of the above systematic survey we present, in Table 2, a summary of the papers considered above.

The popularity in the literature of subgradient optimisation and lagrangean heuristics is clear from that table. In particular note that a common theme nowadays is to use a lagrangean heuristic to generate a heuristic solution to a problem (thereby automatically obtaining an associated quality guarantee) without going to the computational expense of solving the problem optimally.

Table 2: Summary of 1991 papers reviewed

Authors	Subgradient optimisation	Multiplier adjustment	Dual ascent	Lagrangean heuristic	Problem reduction	Tree search
Ahmadi and Matsuo [2]	yes	-	-	yes	-	-
Balas and Saltzman [6]	yes	-	yes	yes	-	yes
Shulman [55]	yes	-	-	yes	-	-
Altinkemer and Gavish [4]	yes	-	-	-	-	-
Noon and Bean [50]	yes	-	-	yes	yes	yes
Ahmadi and Tang [3]	yes	-	-	yes	-	-
Pirkul and Schilling [51]	yes	-	-	yes	-	-
Campbell and Mabert [17]	yes	-	-	yes	-	-
Gavish and Pirkul [33]	yes	-	-	yes	-	-
Domich et al [24]	yes	-	-	yes	-	-
Bard and Bejjani [8]	yes	-	-	yes	-	-
Drexl [25]	-	yes	-	-	-	yes
Cornuejols et al [21]	yes	-	-	yes	-	-
Aboudi et al [1]	-	-	-	-	-	-
Sharma [54]	yes	-	-	yes	-	-
Deckro et al [23]	-	-	-	-	-	-
Lozano et al [48]	-	-	-	yes	-	-
Wright and von Lanzenauer [56]	-	yes	-	yes	-	-
Current and Pirkul [22]	yes	-	-	yes	-	-
Barcelo et al [7]	yes	-	-	yes	yes	yes
Millar and Gunn [49]	yes	-	-	-	-	-

7. Conclusions

In this chapter we have discussed:

- lagrangean relaxation
- lagrangean heuristics
- problem reduction
- subgradient optimisation
- multiplier adjustment
- dual ascent
- tree search

Plainly there are many possible techniques that can be brought to bear on combinatorial optimisation 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 optimisation problems is an art.

However the basic building blocks are clear, namely the types of techniques considered in this, and other, chapters in this book.

In many ways the work that goes on (all around the world) developing algorithms for the solution of combinatorial optimisation problems in 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 optimisation problems, for me at least, 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.

Bibliography

Below we list the references associated with this chapter. These references also include additional reading as follows:

- for the set covering problem see [5,11,14,18,30]
- for computational complexity see [32,45-47,52]
- for lagrangean relaxation see [27,28,34]
- for subgradient optimisation see [9,27,28,34,35,43]
- for multiplier adjustment see [27,28,36,39]
- for dual ascent see [6,16,26,31]
- for lagrangean decomposition see [37,38,44,53]

[1] Aboudi, R., Hallefjord, A. and Jörnsten, K., "A facet generation and relaxation technique applied to an assignment problem with side constraints", European Journal of Operational Research 50 (1991) 335-344.

[2] Ahmadi, R.H. and Matsuo, H., "The line segmentation problem", Operations Research 39 (1991) 42-55.

[3] Ahmadi, R.H. and Tang, C.S., "An operation partitioning problem for automated assembly system design", Operations Research 39 (1991) 824-835.

[4] Altinkemer, K. and Gavish, B., "Parallel savings based heuristics for the delivery problem", Operations Research 39 (1991) 456-469.

[5] Balas, E. and Ho, A., "Set covering algorithms using cutting planes, heuristics, and subgradient optimization: a computational study", Mathematical Programming 12 (1980) 37-60.

[6] Balas, E. and Saltzman, M.J., "An algorithm for the three-index assignment problem", Operations Research 39 (1991) 150-161.

[7] Barcelo, J., Fernandez, E. and Jörnsten, K.O., "Computational results from a new lagrangean relaxation algorithm for the capacitated plant location problem", European Journal of

Operational Research 53 (1991) 38-45.

[8] Bard, J.F. and Bejjani, W.A., "Designing telecommunications networks for the reseller market", Management Science 37 (1991) 1125-1146.

[9] Bazaraa, M.S. and Goode, J.J., "A survey of various tactics for generating lagrangian multipliers in the context of lagrangian duality", European Journal of Operational Research 3 (1979) 322-338.

[10] Beasley, J.E., "A note on solving large p-median problems", European Journal of Operational Research 21 (1985) 270-273.

[11] Beasley, J.E., "An algorithm for set covering problems", European Journal of Operational Research 31 (1987) 85-93.

[12] Beasley, J.E., "An algorithm for solving large capacitated warehouse location problems", European Journal of Operational Research 33 (1988) 314-325.

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

[14] Beasley, J.E., "A lagrangian heuristic for set-covering problems", Naval Research Logistics 37 (1990) 151-164.

[15] Beasley, J.E., "Lagrangean heuristics for location problems", to appear in European Journal of Operational Research.

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

[17] Campbell, G.M. and Mabert, V.A., "Cyclical schedules for capacitated lot sizing with dynamic demands", Management Science 37 (1991) 409-427.

[18] Chan, T.J and Yano, C.A., "A multiplier adjustment approach for

the set partitioning problem", Operations Research 40 (1992) S40-S47.

[19] Christofides, N. and Beasley, J.E., "A tree search algorithm for the p-median problem", European Journal of Operational Research 10 (1982) 196-204.

[20] Christofides, N. and Beasley, J.E., "Extensions to a Lagrangean relaxation approach for the capacitated warehouse location problem", European Journal of Operational Research 12 (1983) 19-28.

[21] Cornuejols, G., Sridharan, R. and Thizy, J.M., "A comparison of heuristics and relaxations for the capacitated plant location problem", European Journal of Operational Research 50 (1991) 280-297.

[22] Current, J. and Pirkul, H., "The hierarchical network design problem with transshipment facilities", European Journal of Operational Research 52 (1991) 338-347.

[23] Deckro, R.F., Winkofsky, E.P., Hebert, J.E. and Gagnon, R.,"A decomposition approach to multi-project scheduling", European Journal of Operational Research 51 (1991) 110-118.

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

[25] Drexl, A., "Scheduling of project networks by job assignment", Management Science 37 (1991) 1590-1602.

[26] Erlenkotter, D., "A dual-based procedure for uncapacitated facility location", Operations Research 26 (1978) 992-1009.

[27] Fisher, M.L., "The lagrangian relaxation method for solving integer programming problems", Management Science 27 (1981) 1-18.

[28] Fisher, M.L., "An applications oriented guide to lagrangian relaxation", Interfaces 15(2) (1985) 10-21.

[29] Fisher, M.L., Jaikumar, R. and Van Wassenhove, L.N., "A multiplier adjustment method for the generalized assignment problem", Management Science 32 (1986) 1095-1103.

[30] Fisher, M.L. and Kedia, P., "Optimal solution of set covering/partitioning problems using dual heuristics", Management Science 36 (1990) 674-688.

[31] Galvao, R.D., "A dual-bounded algorithm for the p-median problem", Operations Research 28 (1980) 1112-1121.

[32] Garey, M.R. and Johnson, D.S., "Computers and intractability: a guide to the theory of NP-completeness", W.H.Freeman, San Francisco, 1979.

[33] Gavish, B. and Pirkul, H., "Algorithms for the multi-resource generalized assignment problem", Management Science 37 (1991) 695-713.

[34] Geoffrion, A.M., "Lagrangean relaxation for integer programming", Mathematical Programming Study 2 (1974) 82-114.

[35] Goffin, J.L., "On the convergence rates of subgradient optimization methods", Mathematical Programming 13 (1977) 329-347.

[36] Guignard, M., "A Lagrangean dual ascent algorithm for simple plant location problems", European Journal of Operational Research 35 (1988) 193-200.

[37] Guignard, M. and Kim, S., "Lagrangean decomposition: a model yielding stronger lagrangean bounds", Mathematical Programming 39 (1987) 215-228.

[38] Guignard, M. and Kim, S., "Lagrangean decomposition for integer

programming: theory and applications", RAIRO 21 (1987) 307-323.

[39] Guignard, M. and Rosenwein, M.B., "An application-oriented guide for designing lagrangean dual ascent algorithms", European Journal of Operational Research 43 (1989) 197-205.

[40] Guignard, M. and Rosenwein, M.B., "An improved dual based algorithm for the generalized assignment problem", Operations Research 37 (1989) 658-663.

[41] Held, M. and Karp, R.M., "The travelling-salesman problem and minimum spanning trees", Operations Research 18 (1970) 1138-1162.

[42] Held, M. and Karp, R.M., "The travelling-salesman problem and minimum spanning trees: Part II", Mathematical Programming 1 (1971) 6-25.

[43] Held, M.H., Wolfe, P. and Crowder, H.D., "Validation of subgradient optimization" Mathematical Programming 6 (1974) 62-88.

[44] Jörnsten, K. and Nasberg, M., "A new Lagrangian relaxation approach to the generalized assignment problem", European Journal of Operational Research 27 (1986) 313-323.

[45] Karp, R.M., "Combinatorics, complexity and randomness", Communications of the ACM 29 (1986) 98-117.

[46] Kolata, G.B., "Solve one and you could solve them all", New Scientist 3rd April 1980, 8-10.

[47] Lewis, H.R. and Papadimitriou, C.H., "The efficiency of algorithms", Scientific American 238(1), January 1978, 96-109.

[48] Lozano, S., Larraneta, J. and Onieva, L., "Primal-dual approach to the single level capacitated lot-sizing problem", European Journal of Operational Research 51 (1991) 354-366.

[49] Millar, H.H. and Gunn, E.A., "Dispatching a fishing trawler fleet in the Canadian Atlantic groundfish industry", European Journal of Operational Research 55 (1991) 148-164.

[50] Noon, C.E. and Bean, J.C., "A lagrangian based approach for the asymmetric generalized travelling salesman problem", Operations Research 39 (1991) 623-632.

[51] Pirkul, H. and Schilling, D.A., "The maximal covering location problem with capacities on total workload", Management Science 37 (1991) 233-248.

[52] Rayward-Smith, V.J., "A first course in computability", (1986) Blackwell Scientific Publications, Oxford.

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

[54] Sharma, R.R.K., "Modelling a fertiliser distribution system", European Journal of Operational Research 51 (1991) 24-34.

[55] Shulman, A. "An algorithm for solving dynamic capacitated plant location problems with discrete expansion sizes", Operations Research 39 (1991) 423-436.

[56] Wright, D. and von Lanzenauer, C.H., "COAL: a new heuristic approach for solving the fixed charge problem - computational results", European Journal of Operational Research 52 (1991) 235-246.

APPENDIX

Binary depth-first tree search code

Below we give a FORTRAN code for a binary depth-first tree search procedure for minimisation problems. This code is generalised in the sense that the user interfaces via his own subroutines - which are:

RECORD which records tree nodes
RECALL which recalls tree nodes

IN which forces a variable into solution (the x=1 branch)
OUT which forces a variable out of solution (the x=0 branch)

BOUND which:
(a) calculates the lower bound for a tree node; and
(b) updates ZUB [the value of the best feasible solution found - an upper bound on the optimal solution to the problem] if an improved feasible solution is found e.g. at a terminal node in the tree, or by a lagrangean heuristic

BRANCH which decides the variable for forward branching (the x=1 branch) at any tree node

FORTRAN code

```
      SUBROUTINE SEARCH(ITREE,LBOUND,N,LBSTART,ZUB)
      INTEGER ITREE(N)
      REAL LBOUND(N),LBSTART,ZUB,ZMAX
C
C DOES THE TREE SEARCH (BINARY, DEPTH-FIRST,
C MINIMISATION)
C
C THIS ROUTINE WORKS OFF THE CONCEPT OF LEVELS
C (DEPTH) IN THE TREE
C
```

```
C ITREE(J) RECORDS THE BRANCH DECISION AT LEVEL J
C ITREE POSITIVE FOR A X=1 BRANCH
C ITREE NEGATIVE FOR A X=0 BRANCH
C
C LBOUND(J) HOLDS THE LOWER BOUND FOR LEVEL J
C
C N IS THE MAXIMUM DEPTH OF THE TREE (TYPICALLY
C THE NUMBER OF VARIABLES IN THE PROBLEM)
C
C LBSTART IS THE LOWER BOUND AT THE ROOT NODE OF
C THE TREE
C
C ZUB IS THE BEST UPPER BOUND KNOWN
C
C RECORD START POINT
C
      CALL RECORD(0)
C
C INITIALISE LEVEL AND NUMBER OF TREE NODES
C
      LEVEL=0
      NTNODES=0
C
  100 CONTINUE
C
C GET BRANCHING VARIABLE
C
      CALL BRANCH(IVAR)
C
      IF(IVAR.EQ.0)WRITE(6,500)
  500 FORMAT(1X,'NO VARIABLE FOR FORWARD BRANCH')
C
C CHECK FOR OK
C
      IF(IVAR.EQ.0.AND.LEVEL.EQ.0)GO TO 350
      IF(IVAR.EQ.0)GO TO 300
C
C HERE OK
C
```



```
      NTNODES=NTNODES+1
      LEVEL=LEVEL+1
C
      ITREE(LEVEL)=+IVAR
C
      WRITE(6,501)LEVEL,IVAR
  501 FORMAT(1X,'LEVEL',I6,' VARIABLE',I6,' INTO TREE')
C
C HERE ADD VARIABLE TO CURRENT SOLUTION
C
      CALL IN(IVAR)
C
  200 CONTINUE
C
C HERE BOUND CALCULATION
C
      CALL BOUND(ZMAX,ZUB)
C
C CHECK LOWER BOUND (ZMAX) AGAINST BEST UPPER
C BOUND (ZUB)
C
      IF(ZMAX.GE.ZUB)GO TO 300
C
C HERE FORWARD
C
C RECORD THE TREE NODE
C
      CALL RECORD(LEVEL)
      LBOUND(LEVEL)=ZMAX
C
      GO TO 100
C
  300 CONTINUE
C
C HERE BACKTRACK
C
      IF(ITREE(LEVEL).LE.0)GO TO 350
C
      IF(LEVEL.EQ.1.AND.LBSTART.GE.ZUB)GO TO 350
```

```
      IF(LEVEL.EQ.1)GO TO 351
C
      IF(LBOUND(LEVEL-1).GE.ZUB)GO TO 350
C
 351  CONTINUE
C
C HERE SET VARIABLE THAT WAS IN TO OUT
C
      CALL RECALL(LEVEL-1)
      IVAR=ITREE(LEVEL)
      ITREE(LEVEL)=-IVAR
C
      CALL OUT(IVAR)
C
      NTNODES=NTNODES+1
C
      WRITE(6,502)LEVEL,IVAR
 502  FORMAT(1X,'LEVEL',I6,' VARIABLE',I6,' OUT OF TREE')
C
      GO TO 200
C
 350  CONTINUE
C
      LEVEL=LEVEL-1
      IF(LEVEL.GT.0)GO TO 300
C
C RECALL LEVEL 0
C
      CALL RECALL(0)
C
      RETURN
      END
```



```
      SUBROUTINE BOUND(ZMAX,ZUB)
      REAL ZMAX,ZUB
C
C CALCULATES THE BOUND AT THE CURRENT TREE NODE
C
C ZMAX IS THE (BEST) LOWER BOUND
C AT THE TREE NODE
C
C ZUB IS UPDATED IF AN IMPROVED FEASIBLE
C SOLUTION IS FOUND
C
.
.
.
      RETURN
      END
```

```
      SUBROUTINE RECORD(J)
C
C RECORDS THE INFORMATION (DATA STRUCTURE) FOR
C THE TREE NODE AT LEVEL J
C
C CAN RECORD ANYTHING ABOUT THE TREE NODE AT
C LEVEL J HERE EXCEPT THE VARIABLES LEVEL, ITREE,
C LBOUND AND ZUB
C
C HERE USE FILE 1
C
      WRITE(1,REC=(J+1))....
C
      RETURN
      END



      SUBROUTINE RECALL(J)
C
C RECALLS THE INFORMATION (DATA STRUCTURE) FOR
C THE TREE NODE AT LEVEL J
C
C HERE USE FILE 1
C
      READ(1,REC=(J+1))....
C
      RETURN
      END
```



```
      SUBROUTINE IN(IVAR)
C
C ALTERS THE DATA STRUCTURE TO SET VARIABLE IVAR
C TO ONE
C
.
.
.
      RETURN
      END



      SUBROUTINE OUT(IVAR)
C
C ALTERS THE DATA STRUCTURE TO SET VARIABLE IVAR
C TO ZERO
C
.
.
.
      RETURN
      END



      SUBROUTINE BRANCH(IVAR)
C
C RETURNS IVAR=0 IF NO BRANCHING VARIABLE
C ELSE IVAR THE VARIABLE TO SET TO ONE
C
.
.
.
      RETURN
      END
```