Integer programming solution methods **J E Beasley**

Introduction

Suppose that we have some problem instance of a combinatorial optimisation 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:

- above the optimal solution value are upper bounds, values which are above the (unknown) optimal solution value
- below the optimal solution value are lower bounds, values which are below the (unknown) optimal solution value.

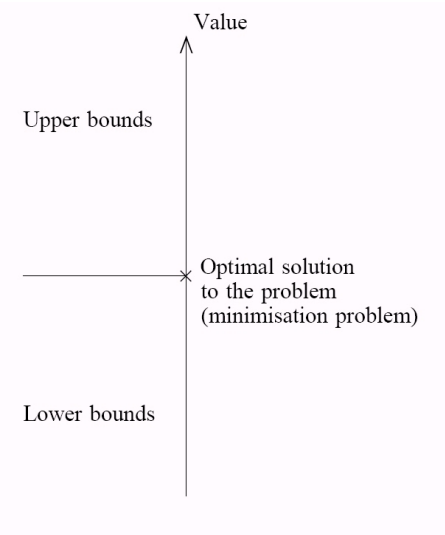

Figure 1

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:

- we like upper bounds that are as close as possible to the optimal solution, i.e. as small as possible
- we like lower bounds that are as close as possible to the optimal solution, i.e. as large as possible.

Upper bounds

Techniques for generating upper bounds are essentially beyond the scope of this course. 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:

- interchange
- metaheuristics:
    - tabu search
    - simulated annealing
    - variable neighbourhood search
    - genetic algorithms (population heuristics).

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

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:

- solved optimally using a standard algorithm (simplex or interior point); or
- solved heuristically (dual ascent).

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

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 course. 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 (optimally) 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):

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

Choosing values for the lagrange multipliers is of key importance in terms of the quality of the lower bound generated (we much prefer lower bounds which are close to the optimal solution). Two general techniques are available here:

- subgradient optimisation; and
- multiplier adjustment.

Preliminaries

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

Problem (P)

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

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

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

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

This linear program can be solved optimally 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.

Lagrangean relaxation

Lagrangean relaxation was developed in the early 1970's with the pioneering work of Held and Karp 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:

- to have chosen some set of constraints in the problem for relaxation; and
- 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 any $\lambda \geq 0$, gives a lower bound on the optimal solution to the original problem P. This can be seen as follows:

The value of 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

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

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

The program after lagrangean relaxation, namely:

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

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 <u>strategic</u> issue, namely why did we choose to relax the set of constraints $Ax \geq b$ when we could equally well have chosen to relax $Bx \geq d$.
- a <u>tactical</u> issue, namely how can we find numerical values for the multipliers.

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

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

This program is called the <u>lagrangean dual program</u>.

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.

<u>Set covering problem</u>

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:

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

the SCP is:

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

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

$$x_j \in (0,1) \qquad j=1,...,n$$

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.

An example SCP (with 3 rows and 4 columns) is:

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

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

Here column 1, of cost 2, covers rows 1 and 2; column 2 of cost 3 covers row 3; column 3 of cost 4 covers rows 1 and 3; column 4 of cost 5 covers rows 2 and 3.

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

(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:

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

subject to $x_j \in (0,1)$ j=1,...,n

i.e.

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

subject to $x_j \in (0,1)$ j=1,...,n

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

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

minimise $\sum_{j=1}^{n} C_jx_j + \sum_{i=1}^{m} \lambda_i$

subject to $x_j \in (0,1)$ j=1,...,n

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

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 2 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.

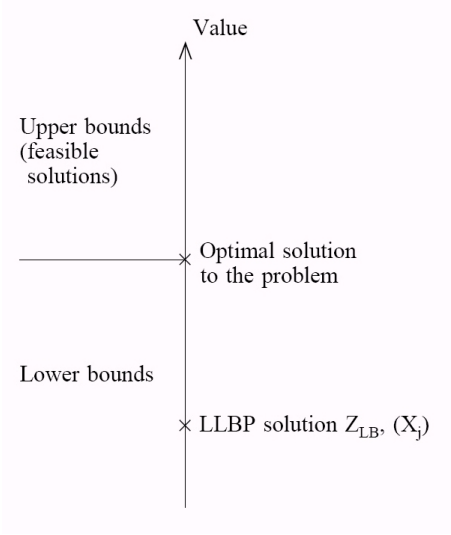

Figure 2

To illustrate the lagrangean relaxation of the SCP given above consider our example SCP:

(c_j) = (2,3,4,5)
(a_{ij}) = | 1 0 1 0 |
| 1 0 0 1 |
| 0 1 1 1 |

Mathematically this example SCP is:

minimise $2x_1 + 3x_2 + 4x_3 + 5x_4$
subject to $x_1 + x_3 \geq 1$
$x_1 + x_4 \geq 1$
$x_2 + x_3 + x_4 \geq 1$
$x_j \in (0,1)$ j=1,...,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:

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,...,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,...,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,...,4

As before, (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

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

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.

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 + λ(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 + λ(b - AX)] i.e. λ(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 [$0 \leq x \leq 1$] then the lagrangean relaxation/lagrangean lower bound program is said to have the integrality property.

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

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

subject to $x_j \in (0,1)$ $j=1,...,n$

with solution

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

It is clear that replacing $x_j \in (0,1)$ $j=1,...,n$ by $0 \leq x_j \leq 1$ $j=1,...,n$ 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.

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 2).

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 integer solution value) 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 integer solution.

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:

- set S=[j | X_j=1 j=1,...,n]
- 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 | a_{ij}=1 j=1,...,n] to S
- S will now be a feasible solution to the original SCP of cost $\sum_{j \in S} c_j$

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:

- S=[1]
- row 3 is the only uncovered row and the minimum cost column covering this row is column 2 so add column 2 to S
- 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, has been that relatively simple lagrangean heuristics can give good quality results.

Deciding lagrange multipliers

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

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

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):

- subgradient optimisation; and
- multiplier adjustment.

We deal with each in turn.

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,...,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:

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

(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

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

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:

- limiting the number of iterations that can be done; or
- 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:

$\lambda_1 = \max(0, 1.5 + 3.6(0)) = 1.5$
$\lambda_2 = \max(0, 1.6 + 3.6(0)) = 1.6$
$\lambda_3 = \max(0, 2.2 + 3.6(1)) = 5.8$

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 (previously 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 $Z_{max}=-\infty$ 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 3 illustrates the situation as we perform subgradient iterations. As shown in that figure we plot the lower bound found at each subgradient iteration on the value line. The best (maximum) of these lower bounds is Z_{max}. This is the lower bound closest to the optimal solution.

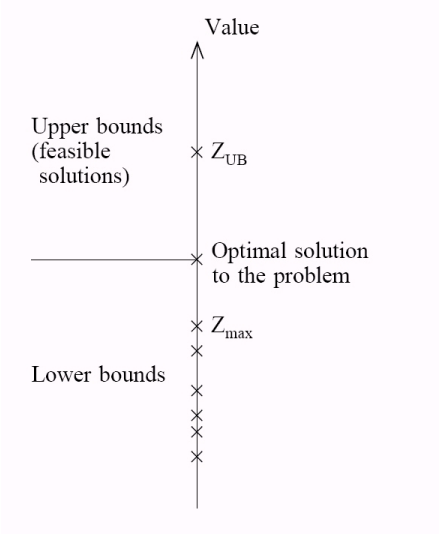

Figure 3

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:

- usually computationally cheap; and
- usually get an increase (or at least no decrease) in the lower bound at each iteration.

The price we pay for this advantage is:

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

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

- increasing λ_i will increase the lower bound obtained from LLBP; and
- the maximum amount (δ) by which we can increase λ_i before the solution to LLBP changes is given by:

  $\delta=\min(C_j \mid a_{ij}=1\ j=1,...,n)$

  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, which were associated with a lower bound of 4.2.

(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 $Z_{LB} = 4.5$, an improvement over the original lower bound of 4.2, as we expect (from the manner in which we designed our multiplier adjustment algorithm to improve the lower bound).

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.

Dual ascent

Dual ascent came to prominence with work on the uncapacitated warehouse (facility) 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:

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

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

- upper bounds, the region above the optimal (integer) solution;
- dual LP feasible solutions, the region below the LP relaxation solution; and
- the gap, the region between the LP relaxation solution and the optimal (integer) solution.

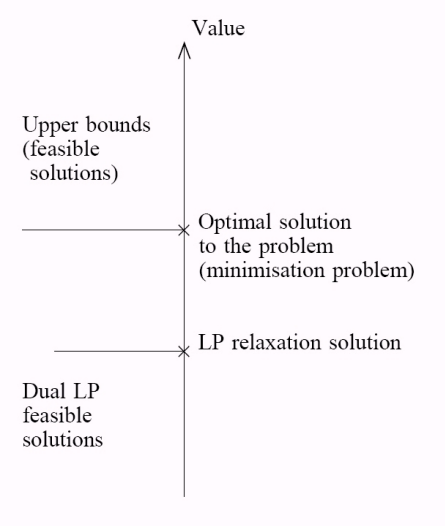

Figure 4

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

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

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

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

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

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

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

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

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

and the dual LP is:

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

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

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

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

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 given above, 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 example investigating 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.

Connections

Consider the two techniques for generating lower bounds that we have given above, namely:
- lagrangean relaxation (with the multipliers being decided by subgradient optimisation or multiplier adjustment); and
- 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:

- the maximum lower bound attainable from LLBP is equal to the value of the LP

relaxation of the original problem.

However it can also be shown that:

- the values of the lagrange multipliers that 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 if the lagrangean relaxation has the integrality property then the optimal lagrange multipliers and the optimal dual variables are the same. This immediately implies that the maximum lower bound attainable from any lagrangean relaxation with the integrality property is equal to the maximum lower bound attainable from any dual ascent algorithm for the problem.

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.

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