Branch and Bound Methods

Stephen Boyd, Arpita Ghosh, and Alessandro Magnani
Notes for EE392o, Stanford University, Autumn 2003

November 1, 2003

Branch and bound algorithms are methods for global optimization in nonconvex problems [LW66, Moo91]. They are nonheuristic, in the sense that they maintain a provable upper and lower bound on the (globally) optimal objective value; they terminate with a certificate proving that the suboptimal point found is ϵ-suboptimal. Branch and bound algorithms can be (and often are) slow, however. In the worst case they require effort that grows exponentially with problem size, but in some cases we are lucky, and the methods converge with much less effort. In these brief notes we describe two typical and simple examples of branch and bound methods.

1 Nonconvex minimization

The material in this section is taken from [BBB91]. The branch and bound algorithm we describe here finds the global minimum of a function $f : \mathbf{R}^m \to \mathbf{R}$ over an m-dimensional rectangle $\mathcal{Q}_{\rm init}$. (Of course, by replacing f by $-f$, the algorithm can also be used to find the global maximum.)

For a rectangle $\mathcal{Q} \subseteq \mathcal{Q}_{\rm init}$ we define

$$\Phi_{\min}(\mathcal{Q}) = \min_{q \in \mathcal{Q}} f(q).$$

Then, the algorithm computes $\Phi_{\min}(\mathcal{Q}_{\rm init})$ to within an absolute accuracy of $\epsilon > 0$, using two functions $\Phi_{\rm lb}(\mathcal{Q})$ and $\Phi_{\rm ub}(\mathcal{Q})$ defined over $\{\, \mathcal{Q} |\, \mathcal{Q} \subseteq \mathcal{Q}_{\rm init}\}$ (which, presumably, are easier to compute than $\Phi_{\min}(\mathcal{Q})$). These two functions satisfy the following conditions.

(R1) $\Phi_{\rm lb}(\mathcal{Q}) \leq \Phi_{\min}(\mathcal{Q}) \leq \Phi_{\rm ub}(\mathcal{Q})$. Thus, the functions $\Phi_{\rm lb}$ and $\Phi_{\rm ub}$ compute a lower and upper bound on $\Phi_{\min}(\mathcal{Q})$, respectively.

(R2) As the maximum half-length of the sides of $\mathcal{Q}$, denoted by $\mathrm{size}(\mathcal{Q})$, goes to zero, the difference between upper and lower bounds *uniformly* converges to zero, *i.e.,*

$$\begin{gathered} \forall\, \epsilon > 0\ \exists\, \delta > 0 \text{ such that} \\ \forall\, \mathcal{Q} \subseteq \mathcal{Q}_{\rm init},\ \ \mathrm{size}(\mathcal{Q}) \leq \delta \Longrightarrow \Phi_{\rm ub}(\mathcal{Q}) - \Phi_{\rm lb}(\mathcal{Q}) \leq \epsilon. \end{gathered}$$

Roughly speaking, then, the bounds $\Phi_{\rm lb}$ and $\Phi_{\rm ub}$ become sharper as the rectangle shrinks to a point.

We now describe the algorithm. We start by computing $\Phi_{\rm lb}(\mathcal{Q}_{\rm init})$ and $\Phi_{\rm ub}(\mathcal{Q}_{\rm init})$. If $\Phi_{\rm ub}(\mathcal{Q}_{\rm init}) - \Phi_{\rm lb}(\mathcal{Q}_{\rm init}) \leq \epsilon$, the algorithm terminates. Otherwise we partition $\mathcal{Q}_{\rm init}$ as a union of subrectangles as $\mathcal{Q}_{\rm init} = \mathcal{Q}_1 \cup \mathcal{Q}_2 \cup \cdots \cup \mathcal{Q}_N$, and compute $\Phi_{\rm lb}(\mathcal{Q}_i)$ and $\Phi_{\rm ub}(\mathcal{Q}_i)$, $i = 1, 2, ..., N$. Then

$$\min_{1\leq i\leq N} \Phi_{\rm lb}(\mathcal{Q}_i) \leq \Phi_{\rm min}(\mathcal{Q}_{\rm init}) \leq \min_{1\leq i\leq N} \Phi_{\rm ub}(\mathcal{Q}_i),$$

so we have new bounds on $\Phi_{\rm min}(\mathcal{Q}_{\rm init})$. If the difference between the new bounds is less than or equal to ϵ, the algorithm terminates. Otherwise, the partition of $\mathcal{Q}_{\rm init}$ is further refined and the bounds updated.

If a partition $\mathcal{Q}_{\rm init} = \cup_{i=1}^N \mathcal{Q}_i$ satisfies size$(\mathcal{Q}_i) \leq \delta$, $i = 1, 2, ..., N$, then by condition (R2) above,

$$\min_{1\leq i\leq N} \Phi_{\rm ub}(\mathcal{Q}_i) - \min_{1\leq i\leq N} \Phi_{\rm lb}(\mathcal{Q}_i) \leq \epsilon;$$

thus a "δ-grid" ensures that $\Phi_{\rm min}(\mathcal{Q}_{\rm init})$ is determined to within an absolute accuracy of ϵ. However, for the "δ-grid", the number of rectangles forming the partition (and therefore the number of upper and lower bound calculations) grows exponentially with $1/\delta$. The branch and bound algorithm applies a heuristic rule for partitioning $\mathcal{Q}_{\rm init}$, which in most cases leads to a reduction of the number of calculations required to solve the problem compared to the δ-grid. The heuristic is this: Given any partition $\mathcal{Q}_{\rm init} = \cup_{i=1}^N \mathcal{Q}_i$ that is to be refined, pick a rectangle $\mathcal{Q}$ from the partition such that $\Phi_{\rm lb}(\mathcal{Q}) = \min_{1\leq i\leq N} \Phi_{\rm lb}(\mathcal{Q}_i)$, and split it into two halves. The rationale behind this rule is that since we are trying to find the minimum of a function, we should concentrate on the "most promising" rectangle. We must emphasize that this is a heuristic, and in the worst case will result in a δ-grid.

In the following description, k stands for the iteration index. $\mathcal{L}_k$ denotes the list of rectangles, L_k the lower bound and U_k the upper bound for $\Phi_{\rm min}(\mathcal{Q}_{\rm init})$, at the end of k iterations.

Branch and bound algorithm

$k = 0$;
$\mathcal{L}_0 = \{\mathcal{Q}_{\rm init}\}$;
$L_0 = \Phi_{\rm lb}(\mathcal{Q}_{\rm init})$;
$U_0 = \Phi_{\rm ub}(\mathcal{Q}_{\rm init})$;
while $U_k - L_k > \epsilon$, {
    pick $\mathcal{Q} \in \mathcal{L}_k$ *such that* $\Phi_{\rm lb}(\mathcal{Q}) = L_k$;
    split $\mathcal{Q}$ *along one of its longest edges into* $\mathcal{Q}_I$ *and* $\mathcal{Q}_{II}$;
    form $\mathcal{L}_{k+1}$ *from* $\mathcal{L}_k$ *by removing* $\mathcal{Q}_k$ *and adding* $\mathcal{Q}_I$ *and* $\mathcal{Q}_{II}$;
    $L_{k+1} := \min_{\mathcal{Q}\in\mathcal{L}_{k+1}} \Phi_{\rm lb}(\mathcal{Q})$;
    $U_{k+1} := \min_{\mathcal{Q}\in\mathcal{L}_{k+1}} \Phi_{\rm ub}(\mathcal{Q})$;
    $k := k + 1$;
}

The requirement that we split the chosen rectangle along a longest edge may seem mysterious at this point. This splitting rule controls the condition number of the rectangles in the partition; see the proof of convergence in §1.1.

At the end of k iterations, U_k and L_k are upper and lower bounds respectively for $\Phi_{\min}(\mathcal{Q}_{\text{init}})$. We prove in §1.1 that if the bounds $\Phi_{\text{lb}}(\mathcal{Q})$ and $\Phi_{\text{ub}}(\mathcal{Q})$ satisfy condition (R2), $U_k - L_k$ is guaranteed to converge to zero, and therefore the branch and bound algorithm will terminate in a finite number of steps.

It is clear that in the branching process described above, the number of rectangles is equal to the number of iterations N. However, we can often eliminate some rectangles from consideration; they may be *pruned* since $\Phi_{\min}(\mathcal{Q}_{\text{init}})$ cannot be achieved in them. This is done as follows. At each iteration:

Eliminate from list $\mathcal{L}_k$ the rectangles $\mathcal{Q} \in \mathcal{L}_k$ that satisfy

$$\Phi_{\text{lb}}(\mathcal{Q}) > U_k.$$

If a rectangle $\mathcal{Q} \in \mathcal{L}_k$ satisfies this condition, then $q \in \mathcal{Q} \Rightarrow f(q) > U_k$; however the minimum of $f(q)$ over $\mathcal{Q}_{\text{init}}$ is *guaranteed* to be less then U_k, and therefore cannot be found in $\mathcal{Q}$.

Though pruning is not necessary for the algorithm to work, it does reduce storage requirements. The algorithm often quickly prunes a large portion of $\mathcal{Q}_{\text{init}}$, and works with only a small remaining subset. The set $\mathcal{L}_k$, the union of the rectangles in the pruned list, acts as an approximation of the set of minimizers of f. In fact, every minimizer of f is guaranteed to be in $\mathcal{L}_k$.

The term *pruning* comes from the following. The algorithm can be viewed as growing a binary tree of rectangles representing the current partition $\mathcal{L}_k$, with the nodes corresponding to rectangles and the children of a given node representing the two halves obtained by splitting it. By removing a rectangle from consideration, we prune the tree.

1.1 Convergence analysis

We now show that the branch and bound algorithm converges in a finite number of steps, provided the bound functions $\Phi_{\text{lb}}(\cdot)$ and $\Phi_{\text{ub}}(\cdot)$ satisfy conditions (R1) and (R2) listed at the beginning of this section. (We will only consider Algorithm I, since the proof for Algorithm II then follows analogously.)

An upper bound on the number of branch and bound iterations

The derivation of an upper bound on the number of iterations of the branch and bound algorithm involves the following steps. We first show that after a large number of iterations k, the partition $\mathcal{L}_k$ must contain a rectangle of small volume. (The volume of a rectangle is defined as the product of the lengths of its sides.) We then show that this rectangle has a small size, and this in turn implies that $U_k - L_k$ is small.

First, we observe that the number of rectangles in the partition $\mathcal{L}_k$ is just k (without pruning, which in any case does not affect the number of iterations). The total volume of these rectangles is $\text{vol}(\mathcal{Q}_{\text{init}})$, and therefore

$$\min_{\mathcal{Q} \in \mathcal{L}_k} \text{vol}(\mathcal{Q}) \leq \frac{\text{vol}(\mathcal{Q}_{\text{init}})}{k}. \tag{1}$$

Thus, after a large number of iterations, at least one rectangle in the partition has small volume.

Next, we show that small volume implies small size for a rectangle in any partition. We define the *condition number* of a rectangle $\mathcal{Q} = \prod_i [l_i, u_i]$ as

$$\mathrm{cond}(\mathcal{Q}) = \frac{\max_i (u_i - l_i)}{\min_i (u_i - l_i)}.$$

We then observe that our splitting rule, which requires that we split rectangles along a longest edge, results in an upper bound on the condition number of rectangles in our partition.

Lemma 1 *For any k and any rectangle $\mathcal{Q} \in \mathcal{L}_k$,*

$$\mathrm{cond}(\mathcal{Q}) \leq \max\{\mathrm{cond}(\mathcal{Q}_{\mathrm{init}}), 2\}. \tag{2}$$

Proof

It is enough to show that when a rectangle $\mathcal{Q}$ is split into rectangles $\mathcal{Q}_1$ and $\mathcal{Q}_2$,

$$\mathrm{cond}(\mathcal{Q}_1) \leq \max\{\mathrm{cond}(\mathcal{Q}), 2\}, \qquad \mathrm{cond}(\mathcal{Q}_2) \leq \max\{\mathrm{cond}(\mathcal{Q}), 2\}.$$

Let $\nu_{\max}$ be the maximum edge length of $\mathcal{Q}$, and $\nu_{\min}$, the minimum. Then $\mathrm{cond}(\mathcal{Q}) = \nu_{\max}/\nu_{\min}$. When $\mathcal{Q}$ is split into $\mathcal{Q}_1$ and $\mathcal{Q}_2$, our splitting rule requires that $\mathcal{Q}$ be split along an edge of length $\nu_{\max}$. Thus, the maximum edge length of $\mathcal{Q}_1$ or $\mathcal{Q}_2$ can be no larger than $\nu_{\max}$. Their minimum edge length could be no smaller than the minimum of $\nu_{\max}/2$ and $\nu_{\min}$, and the result follows. ■

We note that there are other splitting rules that also result in a uniform bound on the condition number of the rectangles in any partition generated. One such rule is to cycle through the index on which we split the rectangle. If $\mathcal{Q}$ was formed by splitting its parent along the ith coordinate, then when we split $\mathcal{Q}$, we split it along the $(i+1)$ modulo m coordinate.

We can bound the size of a rectangle $\mathcal{Q}$ in terms of its volume and condition number, since

$$\begin{aligned}
\mathrm{vol}(\mathcal{Q}) &= \prod_i (u_i - l_i) \\
&\geq \max_i (u_i - l_i) \left(\min_i (u_i - l_i) \right)^{m-1} \\
&= \frac{(2\,\mathrm{size}(\mathcal{Q}))^m}{\mathrm{cond}(\mathcal{Q})^{m-1}} \\
&\geq \left(\frac{2\,\mathrm{size}(\mathcal{Q})}{\mathrm{cond}(\mathcal{Q})} \right)^m .
\end{aligned}$$

Thus,

$$\mathrm{size}(\mathcal{Q}) \leq \frac{1}{2}\,\mathrm{cond}(\mathcal{Q})\mathrm{vol}(\mathcal{Q})^{1/m}. \tag{3}$$

Combining equations (1), (2) and (3) we get

$$\min_{\mathcal{Q}\in\mathcal{L}_k} \text{size}(\mathcal{Q}) \leq \frac{1}{2}\ \max\{\text{cond}(\mathcal{Q}_{\text{init}}), 2\} \left(\frac{\text{vol}(\mathcal{Q}_{\text{init}})}{k}\right)^{1/m}. \tag{4}$$

Thus, for large k, the partition $\mathcal{L}_k$ must contain a rectangle of small size.

Finally, we show that if a partition has a rectangle of small size, the upper and lower bounds cannot be too far apart. More precisely, we show that given some $\epsilon > 0$, there is some N such that $U_N - L_N \leq \epsilon$ for some $N \leq k$.

First, let δ be small enough such that if $\text{size}(\mathcal{Q}) \leq 2\delta$ then $\Phi_{\text{ub}}(\mathcal{Q}) - \Phi_{\text{lb}}(\mathcal{Q}) \leq \epsilon$ (recall requirement (R2) at the beginning of this section). Let k be large enough such that

$$\max\{\text{cond}(\mathcal{Q}_{\text{init}}), 2\} \left(\frac{\text{vol}(\mathcal{Q}_{\text{init}})}{k}\right)^{1/m} \leq 2\ \delta. \tag{5}$$

Then from equation (4), some $\mathcal{Q} \in \mathcal{L}_k$ satisfies $\text{size}(\mathcal{Q}) \leq \delta$. Then the rectangle $\tilde{\mathcal{Q}}$, one of whose halves is $\mathcal{Q}$, must satisfy $\text{size}(\tilde{\mathcal{Q}}) \leq 2\delta$, and therefore

$$\Phi_{\text{ub}}(\tilde{\mathcal{Q}}) - \Phi_{\text{lb}}(\tilde{\mathcal{Q}}) \leq \epsilon.$$

However, since $\tilde{\mathcal{Q}}$ was split at some previous iteration, it must have satisfied $\Phi_{\text{lb}}(\tilde{\mathcal{Q}}) = L_N$ for some $N \leq k$. Thus

$$U_N - L_N \leq \Phi_{\text{ub}}(\tilde{\mathcal{Q}}) - L_N \leq \epsilon,$$

or we have an upper bound on the number of branch and bound iterations.

1.2 An example

In this section we give a simple example, taken from [BBB91]. The function to minimized is a complicated, nonconvex function that arises in control theory, and the lower and upper bounds are found using sophisticated (convex optimization based) relaxation methods, together with control theory. The details don't matter; this example is only meant to show how branch and bound methods work. The specific problem instance considered has 5 variables.

Figure 1 gives a plot of the upper and lower bounds on the optimal value, the pruned volume percentage and the number of active rectangles, versus iterations. We note that the algorithm does not prune any volume at all until about 100 iterations. By about 1000 iterations, 82% of the volume of $\mathcal{Q}_{\text{init}}$ has been pruned and the difference between the upper and lower bounds is down to 0.0155. The algorithm takes about a thousand more iterations to return the solution -0.148, to within an accuracy of 0.001.

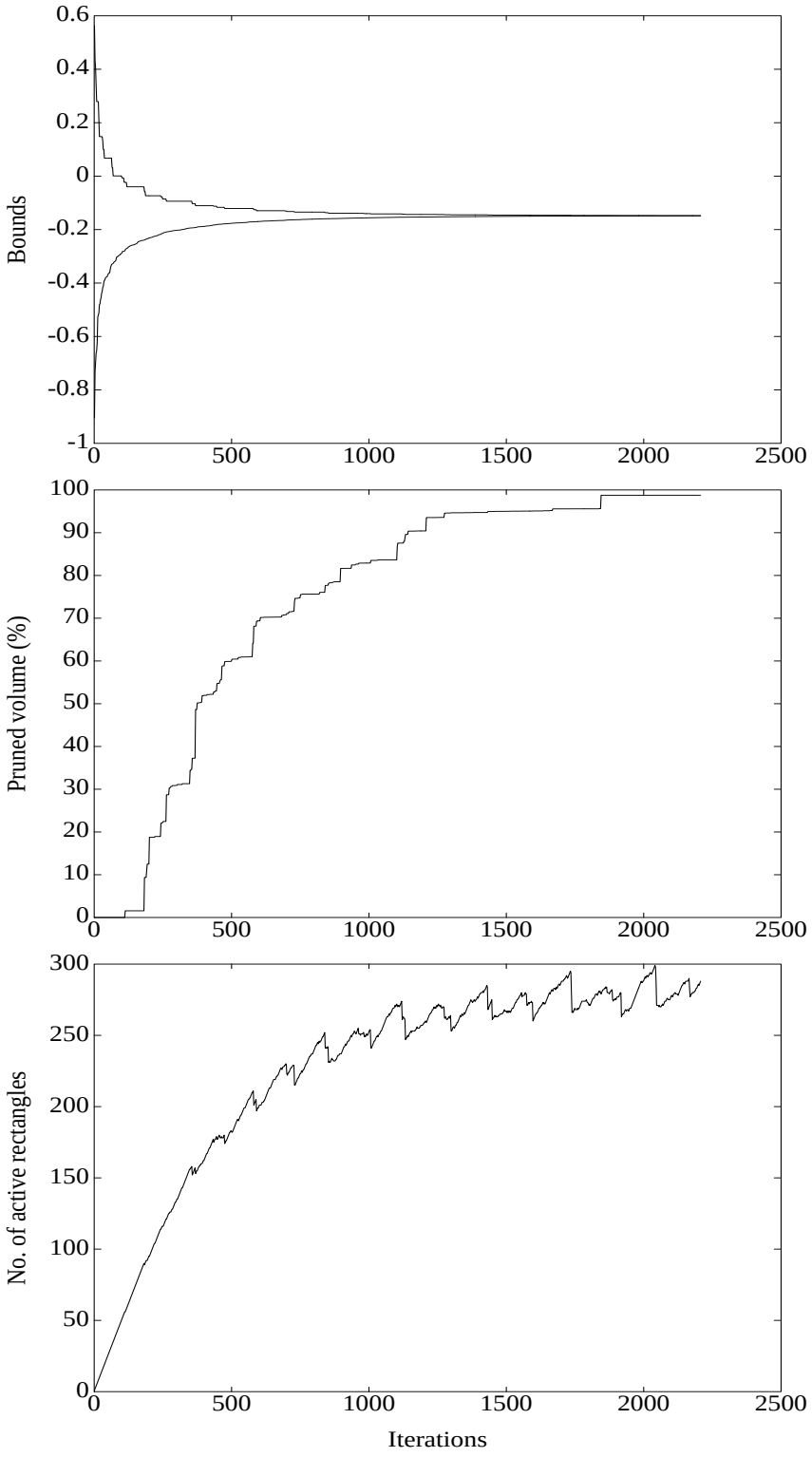

Figure 1: Example: An interval matrix problem.

2 Mixed Boolean-convex problems

In this section we consider another simple example of branch and bound, applied to a combinatorial problem. We consider the following problem:

$$\begin{array}{ll} \text{minimize} & f_0(x,z) \\ \text{subject to} & f_i(x,z) \leq 0, \quad i=1,\ldots,m \\ & z_j \in \{0,1\}, \quad j=1,\ldots,n, \end{array} \tag{6}$$

where x and z are the optimization variables. The variables $z_1,\ldots,z_n$ are, naturally, called *Boolean variables*. We assume the functions f_i, $i=0,\ldots,n$, are convex. This problem is called a *mixed Boolean convex problem*. We denote the optimal value of this problem as $p^\star$.

One way to solve this problem is by exhaustive search. That is, we solve 2^n convex optimization problems, one for each possible value of the (Boolean) vector z, and then choose the smallest of these optimal values. This involves solving a number of convex problems that is exponential in the size of the variable z. For n more than 30 or so, this is clearly not possible.

We will use a branch and bound method to solve this problem. In the worst case, we end up solving the 2^n convex problems, i.e., carrying an exhaustive search. But with luck, this does not occur.

The *convex relaxation*

$$\begin{array}{ll} \text{minimize} & f_0(x,z) \\ \text{subject to} & f_i(x,z) \leq 0, \quad i=1,\ldots,m \\ & 0 \leq z_j \leq 1, \quad j=1,\ldots,n, \end{array} \tag{7}$$

with (continuous) variables x and z, is convex, and so easily solved. Its optimal value, which we denote L_0, is a lower bound on the optimal value of (6). This lower bound can be $+\infty$ (in which case the original problem is surely infeasible) or $-\infty$. We can also get an upper bound on $p^\star$ using the relaxation. For example, we can take the solution of the relaxed problem, and then round each of the variables z_i to 0 or 1. This upper bound can be $+\infty$, if the rounded solution isn't feasible. We'll denote this upper bound by U_0. Of course, if we have $U_0 - L_0 \leq \epsilon$, the required tolerance, we can quit.

Now we are going to branch. Pick any index k, and form two problems: The first problem is

$$\begin{array}{ll} \text{minimize} & f_0(x,z) \\ \text{subject to} & f_i(x,z) \leq 0, \quad i=1,\ldots,m \\ & z_j \in \{0,1\}, \quad j=1,\ldots,n \\ & z_k = 0 \end{array}$$

and the second problem is

$$\begin{array}{ll} \text{minimize} & f_0(x,z) \\ \text{subject to} & f_i(x,z) \leq 0, \quad i=1,\ldots,m \\ & z_j \in \{0,1\}, \quad j=1,\ldots,n \\ & z_k = 1. \end{array}$$

In other words, we fix the value of z_k to 0 in the first problem, and 1 in the second. We call these subproblems of the original, since they can be thought of as the same problem, with variable eliminated or fixed. Each of these subproblems is also a mixed Boolean convex problem, with $n-1$ Boolean variables. The optimal value of the original problem is clearly the smaller of the optimal values of these two subproblems.

We now solve the two convex relaxations of these subproblems with $z_k = 0$ and $z_i = 1$, to obtain a lower and upper bound on the optimal value of each subproblem. We'll denote these as $\tilde{L}, \tilde{U}$ (for $z_k = 0$) and $\bar{L}, \bar{U}$ (for $z_k = 1$). Each of these two lower bounds must be larger than L_0, i.e., $\min\{\tilde{L}, \bar{L}\} \geq L_0$. We can also assume, without loss of generality, that $\min\{\tilde{U}, \bar{U}\} \leq U_0$. From these two sets of bounds, we obtain the following bounds on $p^\star$:

$$L_1 = \min\{\tilde{L}, \bar{L}\} \leq p^\star \leq U_1 = \min\{\tilde{U}, \bar{U}\}.$$

By the inequalities above, we have $U_1 - L_1 \leq U_0 - L_0$.

At the next step, we choose either of the subproblems, and then split it, by choosing an index (not equal to k, the index used to split the original problem). We solve the relaxations for the split subproblems (which have $n-2$ Boolean variables), and obtain lower and upper bounds for each.

At this point we have formed a partial Boolean tree of subproblems. The root is the original problem; the first split yields two children subproblems, one with $z_k = 0$ and one with $z_k = 0$. The second iteration yields another two children of one of the original children. We continue is this way. At each iteration, we choose a leaf node (which corresponds to a subproblem, with some of the Boolean variables fixed to particular values), and split it, by fixing a variable that is not fixed in the parent problem. An edge in the tree corresponds to the value 0 or 1 of a particular variable z_i. At the root node of the tree, the values of none of the z_i are specified. A node at depth k in the tree corresponds to a subproblem in which k of the Boolean variables have fixed values. For each node, we have an upper and lower bound on the optimal value of the subproblem. After k iterations, we have a tree with exactly k leaves.

The minimum of the lower bounds, over all the leaf nodes, gives a lower bound on the optimal value $p^\star$; similarly, the minimum of the upper bounds, over all the leaf nodes, gives an upper bound on the optimal value $p^\star$. We refer to these lower and upper bounds as L_k and U_k, respectively. We always have $U_{k+1} - L_{k+1} \leq U_k - L_k$; we can terminate the algorithm when $U_k - L_k \leq \epsilon$.

Proving convergence of the algorithm is trivial: it must terminate in fewer than 2^n steps. To see this, note that if a leaf has depth n, it means that all the Boolean variables are fixed in the subproblem, so by solving the convex relaxation we get the exact solution. In other words, for any leaf of depth n, we have $U = L$. The worst thing that can happen is that we develop a complete binary tree, to depth n, which requires 2^n steps, at which point every subproblem lower and upper bound is equal, and therefore the algorithm terminates. This is nothing more than exhaustive search.

At any point in the algorithm we have an upper bound U on $p^\star$. If any node has a lower bound that is more than U, we can *prune* it, i.e., remove it from the tree.

The choice of which leaf to split at a given stage in the algorithm is arbitrary. Several heuristics are used. One is to split the leaf corresponding to the smallest lower bound, since

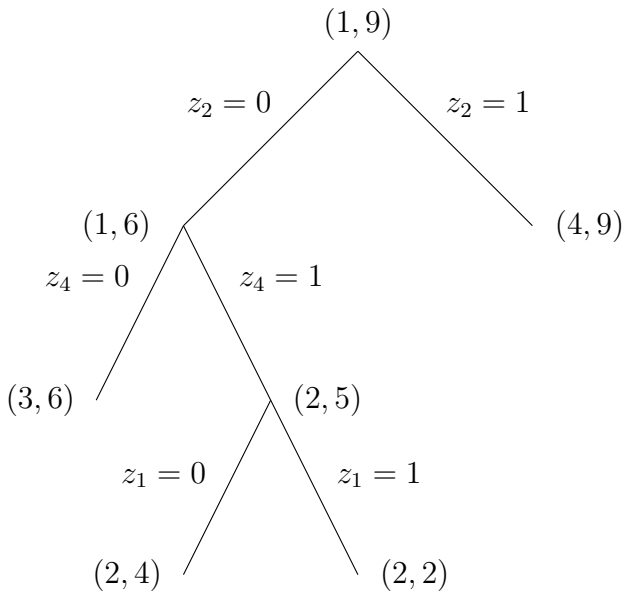

Figure 2: Branch and bound algorithm for mixed convex Boolean problem.

we might expect to find the optimal value in that subtree, and in any case splitting this leaf will lead to an improvement in the global lower bound. The same can be said for the choice of index to use to split a leaf. It can be any index corresponding to a Boolean variable that has not yet been fixed. One heuristic is to split along a variable k for which to $z_k = 0$ or 1 in the relaxation; the hope is that this might allow us to discard the whole subtree starting from the other value. If many of the relaxed values are 0 or 1, we can choose the one with the largest associated Lagrange multiplier. (The idea here is that this is the relaxed variable that is 0 and 1, and has the highest pressure to stay there.)

We illustrate the algorithm with a hypothetical example, with 4 Boolean variables, shown in figure 2. Each node shows the lower and upper bound, and each edge shows the variable that is fixed. The original problem is shown at the root of the tree. The original lower and upper bounds are 1 and 9, respectively. On the first iteration, we decide to split on variable z_2. At this point our global lower bound is 1, and our upper bound is 6. We then split the child with $z_2 = 0$, using index 4. This results in a lower bound of 2 and upper bound 5. In the next iteration, we split the subproblem associated with $z_2 = 0$, $z_4 = 1$, splitting along variable z_1. After this iteration, the algorithm converges: we find that the lower and upper bounds are both 2. In this example, we needed to solve 7 convex problems. Exhaustive search would have required solving $2^4 = 16$ convex problems.

A simple numerical example

We consider the problem

$$\begin{array}{ll} \text{minimize} & c^T z \\ \text{subject to} & d^T z \leq -1, \\ & z_j \in \{0, 1\}, \quad j = 1, \ldots, n, \end{array}$$

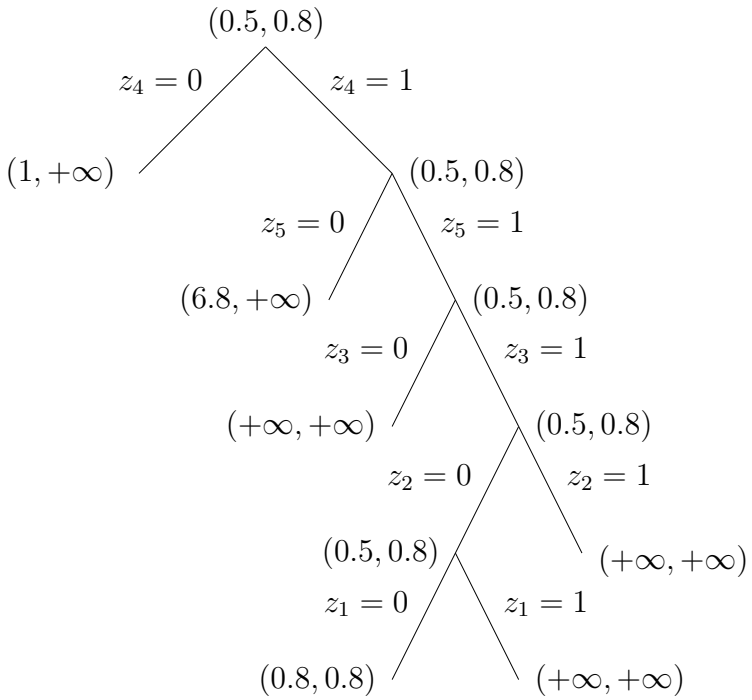

Figure 3: Branch and bound algorithm for Boolean problem.

where

$$c = (1.1, -2.2, 3.4, -3.7, 6), \quad d = (-1.1, -2.2, -3.4, 3.2, 5).$$

In fact, Boolean linear programs of this form (one inequality constraint) are easily solved; we use this example just to illustrate branch and bound. Figure 3 shows the steps of the branch and bound algorithm. As we can see the algorithm terminates in 10 steps, which is fewer than the 32 steps that would be required for exhaustive search.

For this specific problem we pick the leaf of the tree with the lowest lower bound to compute the next iteration of the algorithm. Based on the solution of the relaxation at that node we select the variable we fix at the next iteration: We fix the variable whose solution in the relaxed problem is closest to either 0 or 1. If more than one of the variables is equal to 1 or 0, we select among these one whose associated dual variable is largest. The idea behind this heuristic is to fix the variable that, based on the solution of the relaxation, seems to be more likely equal to be 1 or 0 for the optimal solution.

References

[BBB91] V. Balakrishnan, S. Boyd, and S. Balemi. Branch and bound algorithm for computing the minimum stability degree of parameter-dependent linear systems. *Int. J. of Robust and Nonlinear Control*, 1(4):295–317, October–December 1991.

[LW66] E. L. Lawler and D. E. Wood. Branch-and-bound methods: A survey. *Operations Research*, 14:699–719, 1966.

[Moo91] R. E. Moore. Global optimization to prescribed accuracy. *Computers and Mathematics with Applications*, 21(6/7):25–39, 1991.