Branch and Bound Methods

Stephen Boyd and Jacob Mattingley
Notes for EE364b, Stanford University, Winter 2006-07

April 1, 2018

Branch and bound algorithms are methods for global optimization in nonconvex problems [?, ?]. 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 notes we describe two typical and simple examples of branch and bound methods, and show some typical results, for a minimum cardinality problem.

1 Unconstrained nonconvex minimization

The material in this section is taken from [?]. 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 $\mathcal{Q}_{\rm init}$, to within some prescribed accuracy ϵ. We let $f^\star$ denote the optimal value, *i.e.*, $f^\star = \inf_{x\in\mathcal{Q}_{\rm init}} f(x)$. For a rectangle $\mathcal{Q} \subseteq \mathcal{Q}_{\rm init}$ we define

$$\Phi_{\rm min}(\mathcal{Q}) = \inf_{x\in\mathcal{Q}} f(x),$$

so $f^\star = \Phi_{\rm min}(\mathcal{Q}_{\rm init})$.

The algorithm will use two functions $\Phi_{\rm lb}(\mathcal{Q})$ and $\Phi_{\rm ub}(\mathcal{Q})$ defined for any rectangle $\mathcal{Q} \subseteq \mathcal{Q}_{\rm init}$. These functions must satisfy the following conditions. First, they are lower and upper bounds on $\Phi_{\rm min}(\mathcal{Q})$, respectively: for any $\mathcal{Q} \subseteq \mathcal{Q}_{\rm init}$,

$$\Phi_{\rm lb}(\mathcal{Q}) \le \Phi_{\rm min}(\mathcal{Q}) \le \Phi_{\rm ub}(\mathcal{Q}).$$

The second condition is (roughly) that the bounds become tight as the rectangle shrinks to a point. More precisely, 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.*,

$$\forall\epsilon > 0\ \exists\delta > 0\ \forall\mathcal{Q} \subseteq \mathcal{Q}_{\rm init},\quad \mathrm{size}(\mathcal{Q}) \le \delta \Longrightarrow \Phi_{\rm ub}(\mathcal{Q}) - \Phi_{\rm lb}(\mathcal{Q}) \le \epsilon.$$

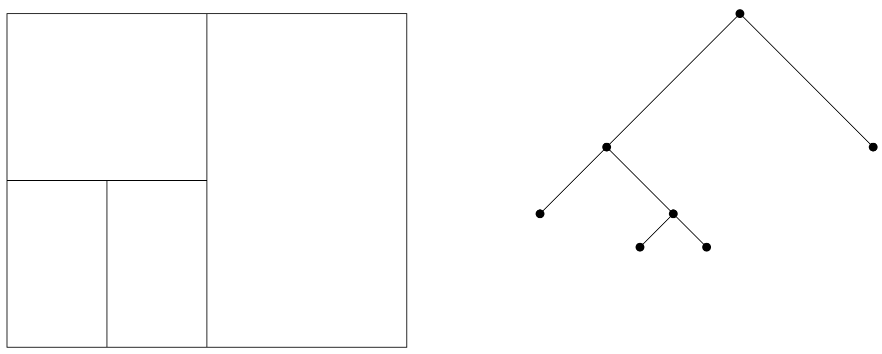

Figure 1: Branch and bound example in $\mathbf{R}^2$, after 3 iterations. The partition of the original rectangle is shown at left; the associated binary tree is shown at right.

We mention a third condition, which is not needed to prove convergence of the branch and bound algorithm, but is needed in practice: The functions $\Phi_{\rm ub}(\mathcal{Q})$ and $\Phi_{\rm lb}(\mathcal{Q})$ should be cheap to compute.

We now describe the algorithm. We start by computing

$$L_1 = \Phi_{\rm lb}(\mathcal{Q}_{\rm init}), \qquad U_1 = \Phi_{\rm ub}(\mathcal{Q}_{\rm init}),$$

which are lower and upper bounds on $f^\star$, respectively. If $U_1 - L_1 \leq \epsilon$, the algorithm terminates. Otherwise we partition $\mathcal{Q}_{\rm init}$ into two rectangles $\mathcal{Q}_{\rm init} = \mathcal{Q}_1 \cup \mathcal{Q}_2$, and compute $\Phi_{\rm lb}(\mathcal{Q}_i)$ and $\Phi_{\rm ub}(\mathcal{Q}_i)$, $i = 1, 2$. Then we have new lower and upper bounds on $f^\star$,

$$L_2 = \min\{\Phi_{\rm lb}(\mathcal{Q}_1), \Phi_{\rm lb}(\mathcal{Q}_2)\} \leq \Phi_{\rm min}(\mathcal{Q}_{\rm init}) \leq U_2 = \min\{\Phi_{\rm ub}(\mathcal{Q}_1), \Phi_{\rm ub}(\mathcal{Q}_2)\}.$$

If $U_2 - L_2 \leq \epsilon$, the algorithm terminates. Otherwise, we partition one of $\mathcal{Q}_1$ and $\mathcal{Q}_2$ into two rectangles, to obtain a new partition of $\mathcal{Q}_{\rm init}$ into three rectangles, and compute $\Phi_{\rm lb}$ and $\Phi_{\rm ub}$ for these new rectangles. We update the lower bound L_3 as the minimum over the lower bounds over the partition of $\mathcal{Q}_{\rm init}$, and similarly for the upper bound U_3.

At each iteration we split one rectangle into two, so after k iterations (counting the evaluation of $\Phi_{\rm lb}(\mathcal{Q}_{\rm init})$ and $\Phi_{\rm ub}(\mathcal{Q}_{\rm init})$ as the first one), we have a partition of $\mathcal{Q}_{\rm init}$ into k rectangles, $\mathcal{Q}_{\rm init} = \cup_{i=1}^k \mathcal{Q}_i$. This partition can be represented by a partially filled binary tree, with the root node corresponding to the original rectangle, and the children of a node corresponding to the subrectangles obtained by partitioning the parent rectangle.

This is illustrated in figure 1. The original rectangle corresponds to the root of the tree. We first partition $\mathcal{Q}_{\rm init}$ vertically. At the next iteration, we horizontally partition the lefthand child; at the third iteration, we vertically partition the bottom child. After three iterations, we have a partition of the original rectangle into 4 rectangles, which correspond to the leaves in the tree.

The associated lower and upper bounds on $f^\star$ are

$$L_k = \min_{i=1,\ldots,k} \Phi_{\rm lb}(\mathcal{Q}_i), \qquad U_k = \min_{i=1,\ldots,k} \Phi_{\rm ub}(\mathcal{Q}_i).$$

We can assume that the lower and upper bounds of the pair of rectangles obtained by splitting are no worse than the lower and upper bounds of the rectangle they were formed from. This implies that L_k is nondecreasing, and U_k is nonincreasing.

To fully specify the algorithm, we need to give the rule for choosing which rectangle to split at each step, and we have to specify which edge along which the rectangle is to be split. One standard method for choosing the rectangle in the current partition to be split is to choose one with the smallest lower bound, *i.e.*, a rectangle that satisfies $\Phi_{\rm lb}(\mathcal{Q}) = L_k$. Once we choose the rectangle to split, we split it along any of its longest edges.

The algorithm is summarized below.

```
$k = 0$;
$\mathcal{L}_0 = \{\mathcal{Q}_{\rm init}\}$;
$L_0 = \Phi_{\rm lb}(\mathcal{Q}_{\rm init})$;
$U_0 = \Phi_{\rm ub}(\mathcal{Q}_{\rm init})$;
while $U_k - L_k > \epsilon$, {
        pick $\mathcal{Q} \in \mathcal{L}_k$ for which $\Phi_{\rm lb}(\mathcal{Q}) = L_k$;
        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 controls the condition number of the rectangles in the partition, which in turn allows us to prove convergence.

As the algorithm proceeds, we can eliminate some rectangles from consideration; they can be *pruned*, since $\Phi_{\rm min}(\mathcal{Q}_{\rm init})$ cannot be achieved in them. This is done as follows. At each iteration, we eliminate from the list $\mathcal{L}_k$ any rectangles that satisfy $\Phi_{\rm lb}(\mathcal{Q}) > U_k$, since every point in such a rectangle is worse than the current upper bound on $f^\star$.

Pruning does not affect the algorithm, since any rectangle pruned would never be selected later for further splitting. But pruning can reduce storage requirements. With pruning, the union of the list of current rectangles can be regarded as the set of possible minimizers of f. The number of active rectangles, and their total volume, give some measure of algorithm progress.

The term *pruning* comes from the following. The algorithm can be viewed as developing a partially filled 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. The leaves of the tree give the current partition. By removing a rectangle from consideration, we prune the tree.

1.1 Convergence analysis

We first show that after a large enough number of iterations, the partition $\mathcal{L}_k$ must contain a rectangle of small volume. We then show that this rectangle has small size, and this in turn implies that $U_k - L_k$ is small.

The number of rectangles in the partition $\mathcal{L}_k$ is k, and the total volume of these rectangles is $\mathrm{vol}(\mathcal{Q}_{\mathrm{init}})$, so

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

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

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

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: for any rectangle $\mathcal{Q} \in \mathcal{L}_k$,

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

To prove this it is enough to show that when a rectangle $\mathcal{Q}$ is split into rectangles $\mathcal{Q}_1$ and $\mathcal{Q}_2$, we have

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

Let $\nu_{\max}$ be the maximum edge length of $\mathcal{Q}$, and let $\nu_{\min}$ be the minimum edge length of $\mathcal{Q}$, so $\mathrm{cond}(\mathcal{Q}) = \nu_{\max}/\nu_{\min}$. When $\mathcal{Q}$ is split into $\mathcal{Q}_1$ and $\mathcal{Q}_2$, the maximum edge length of the children is no more than $\nu_{\max}$. The minimum edge length of the children, by our splitting rule, is no smaller than the minimum of $\nu_{\max}/2$ and $\nu_{\min}$. It follows that the condition number of the children is no more than the maximum of 2 and $\mathrm{cond}(\mathcal{Q})$.

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,

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

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

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

Thus the minimum size of the rectangles in the partition $\mathcal{L}_k$ converges to zero.

By assumption 2, there exists a δ such that any rectangle of size δ or smaller satisfies $\Phi_{\text{ub}}(\mathcal{Q}) - \Phi_{\text{lb}}(\mathcal{Q}) \leq \epsilon$. Take k large enough that at least one rectangle in the partition has size not exceeding δ. Then at least one rectangle, say, $\mathcal{Q}$, in partition $\mathcal{L}_k$ satisfies $\Phi_{\text{ub}}(\mathcal{Q}) - \Phi_{\text{lb}}(\mathcal{Q}) \leq \epsilon$. It follows than when this rectangle was added to the list, the algorithm should have terminated.

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:

$$\begin{array}{ll} \text{minimize} & f_0(x,z) \\ \text{subject to} & f_i(x,z) \leq 0, \quad i = 1,\dots,m \\ & z_j \in \{0,1\}, \quad j = 1,\dots,n, \end{array} \tag{5}$$

where x and z are the optimization variables. The variables $z_1, \dots, z_n$ are, naturally, called *Boolean variables*. We assume the functions f_i, $i = 0, \dots, n$, 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*

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

with (continuous) variables x and z, is convex, and therefore easily solved. Its optimal value, which we denote L_1, is a lower bound on the optimal value of (5). 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. The simplest method is to round each of the relaxed Boolean variables z_i to 0 or 1. A more sophisticated approach is to first round each of the relaxed Boolean variables z_i to 0 or 1, and then, with these values of z_i fixed, solve the resulting convex problem in the variable x.

The upper bound obtained from either of these methods can be $+\infty$, if this method fails to produce a feasible point. (The upper bound can also be $-\infty$, in which case the original problem is surely unbounded below.) We'll denote this upper bound by U_1. Of course, if we have $U_1 - L_1 \leq \epsilon$, 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 one 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 the smaller of the optimal values of the two subproblems.

We now solve the two convex relaxations of these subproblems 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 at least as large as L_1, *i.e.*, $\min\{\tilde{L}, \bar{L}\} \geq L_1$. We can also assume, without loss of generality, that $\min\{\tilde{U}, \bar{U}\} \leq U_1$. From these two sets of bounds, we obtain the following bounds on $p^\star$:

$$L_2 = \min\{\tilde{L}, \bar{L}\} \leq p^\star \leq U_2 = \min\{\tilde{U}, \bar{U}\}.$$

By the inequalities above, we have $U_2 - L_2 \leq U_1 - L_1$.

At the next step, we can choose to split either of the subproblems. We need to split on 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 binary tree of subproblems. The root is the original problem; the first split yields two child subproblems, one with $z_k = 0$ and the other with $z_k = 1$. 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, none of the values 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. The global lower bound L_k is nondecreasing, and the global upper bound U_k is nonincreasing, so 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, with $U_k = L_k$, in fewer than 2^n iterations. 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 iterations, at which point every subproblem lower and upper bound is equal, and therefore the algorithm terminates. This is nothing more than exhaustive search.

We can carry out pruning, to save memory (but not time), as the algorithm progresses. At iteration k we have the upper bound U_k on $p^\star$. Any node in the tree that has a lower bound larger than U can be pruned, *i.e.*, removed from the tree, since all points corresponding to that node are worse than the current best point found.

The choice of which leaf to split at a given stage in the algorithm is somewhat arbitrary. The primary strategy is to split the leaf corresponding to the smallest lower bound. This means we always get an improvement in the global lower bound. The drawback of this scheme is that in some situations (such as the cardinality example in the next section) we may achieve a tight lower bound before getting any optimal feasible points.

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 $|z_k^\star - 1/2|$ is smallest, where $z_k^\star$ is the optimal point found in the relaxation. This corresponds to splitting a variable that is most 'undecided' (between 0 and 1) in the relaxation. In some problems, the opposite heuristic works well: we split a variable that is most 'decided'. To do this, we choose k for which $|z_k^\star - 1/2|$ is maximum. In most cases, many of the variables $z_k^\star$ will be 0 or 1; among these, 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.)

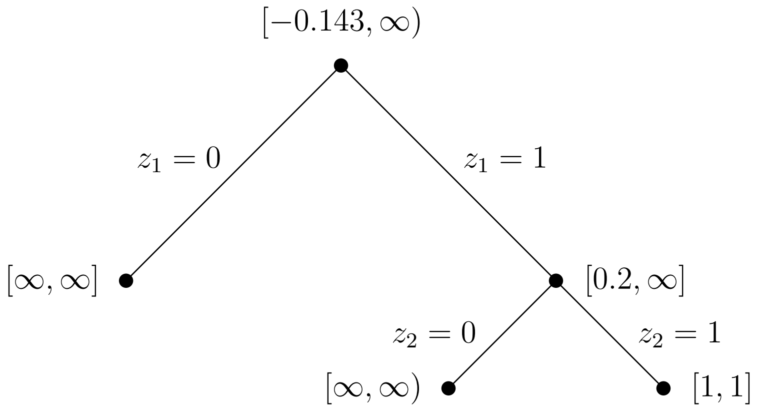

Figure 2: Simple three variable example.

2.1 Small example

We illustrate the algorithm with a small example, the Boolean LP

$$\begin{array}{ll} \text{minimize} & 2z_1 + z_2 - 2z_3 \\ \text{subject to} & 0.7z_1 + 0.5z_2 + z_3 \geq 1.8 \\ & z_i \in \{0, 1\}, \quad i = 1, 2, 3, \end{array}$$

with variables z_1, z_2, and z_3. This problem is small enough to solve by exhaustive enumeration. Out of 8 possible 3-dimensional Boolean vectors, just one is feasible. This optimal point is $z = (1, 1, 1)$.

Figure 2 shows the binary tree corresponding to the branch and bound algorithm, which terminates in 2 iterations, after 5 LP relaxations are solved. Each node shows the lower and upper bound, and each edge shows the variable that is fixed, and its value. At the first node we solve the relaxed problem and obtain the lower bound $L_1 = -0.143$. Rounding the $z_i^\star$ found in the relaxation does not, however, yield a feasible solution, so the upper bound is $U_1 = \infty$.

Setting $z_1 = 0$ we find that the relaxed problem is infeasible, so we have $L = \infty$ (and therefore $U = \infty$). At this point we have determined that, if there is any solution, it must satisfy $z_1 = 1$. Now we explore the $z_1 = 1$ branch. Solving the associated relaxed LP we obtain the new lower bound 0.2, an improvement over the previous global lower bound -0.143. Once again, when we round $z_i^\star$, we get an infeasible point, so the upper bound at this node is still ∞. At this point, two iterations into the algorithm, having solved a total of 3 LPs, we find that $L_2 = 0.2$ and $U_2 = \infty$. (This means that so far, we are not even sure the problem is feasible.)

Now we split on the variable z_2. Setting $z_2 = 1$ we get a relaxed optimal value of 1. In the relaxed problem, we have $z_3^\star = 1$, so (even without rounding) we obtain a feasible point with objective 1. Thus, the lower and upper bounds at this node are both 1. When we set $z_2 = 0$, the relaxed problem is infeasible. For this node, then, we have $L = U = \infty$.

After this third iteration we have

$$L_3 = \min\{\infty, \infty, 1\} = 1, \qquad U_3 = \min\{\infty, \infty, 1\} = 1,$$

which means we have solved the problem globally.

Of course for a simple problem like this we could have simply checked the 8 possible Boolean vectors for feasibility, and evaluated the objective function of those that are feasible. This example was meant only to illustrate the branch and bound method.

3 Minimum cardinality example

We consider the problem of finding the sparsest solution of a set of linear inequalities,

$$\begin{array}{ll} \text{minimize} & \mathbf{card}(x) \\ \text{subject to} & Ax \preceq b. \end{array} \tag{7}$$

This problem is equivalent to the mixed Boolean-LP

$$\begin{array}{ll} \text{minimize} & \mathbf{1}^T z \\ \text{subject to} & L_i z_i \leq x_i \leq U_i z_i, \quad i = 1, \ldots, n \\ & Ax \preceq b \\ & z_i \in \{0, 1\}, \quad i = 1, \ldots, n \end{array} \tag{8}$$

with variables x and z, and where L and U are, respectively, lower and upper bounds on x. We can find the lower bounds L_i by solving the n LPs

$$\begin{array}{ll} \text{minimize} & x_i \\ \text{subject to} & Ax \preceq b, \end{array}$$

for $i = 1, \ldots, n$. (The optimal values of these LPs give $L_1, \ldots, L_n$.) We can find the upper bounds $U_1, \ldots, U_n$ by solving the n LPs

$$\begin{array}{ll} \text{maximize} & x_i \\ \text{subject to} & Ax \preceq b \end{array}$$

for $i = 1, \ldots, n$.

If any L_i is positive, then any feasible x satisfies $x_i > 0$, and it follows that $z_i = 1$. In this case, we can remove the variable z_i (since we already know its value). In a similar way, if any U_i is negative, then we must have $z_i = 1$, and we can simple remove z_i. So we'll assume that $L_i \leq 0$ and $U_i \geq 0$.

We'll solve the mixed Boolean-convex problem (8) using the general method described in §2. The relaxation is

$$\begin{array}{ll} \text{minimize} & \mathbf{1}^T z \\ \text{subject to} & L_i z_i \leq x_i \leq U_i z_i, \quad i = 1, \ldots, n \\ & Ax \preceq b \\ & 0 \leq z_i \leq 1, \quad i = 1, \ldots, n \end{array} \tag{9}$$

which is equivalent to

$$\begin{array}{ll} \text{minimize} & \sum_{i=1}^{n}\left((1/U_i)(x_i)_+ + (-1/L_i)(x_i)_-\right) \\ \text{subject to} & Ax \preceq b \end{array}$$

(assuming $L_i < 0$ and $U_i > 0$, for $i = 1, \ldots, n$). The objective here is an asymmetric weighted ℓ_1-norm. Minimizing the ℓ_1-norm over a convex set is a well known heuristic for finding a sparse solution. We can improve on the lower bound found by the relaxation. Since $\mathbf{card}(x)$ is integer valued, if we know $\mathbf{card}(x) \geq l$, we know that $\mathbf{card}(x) \geq \lceil l \rceil$, the smallest integer greater than or equal to l. (For example, if we know that $\mathbf{card}(x) \geq 23.4$, then in fact we know that $\mathbf{card}(x) \geq 24$.)

We find an upper bound on the mixed Boolean-LP (8) by simply recording the cardinality of the x obtained in the relaxed problem (9).

At each iteration we split a node with the lowest lower bound. This will be the optimal strategy if we find an optimal point before proving the lower bound. Less clear is the choice of variable to split; empirical results suggest that splitting along the 'most ambiguous' variable in the relaxation, *i.e.*, z_j, where $j = \text{argmin}_i(|z_i - 1/2|)$, is an efficient strategy.

3.1 Numerical results

A small example

We first give the results of the branch and bound algorithm for a randomly generated problem of the form (7), with $x \in \mathbf{R}^{30}$ and 100 constraints. It takes 124 iterations, and the solution of a total of 309 LPs (including the 60 LPs required to compute the lower and upper bounds on each variable) to find that the minimum cardinality is 19.

Figure 3 shows the tree generated by the algorithm at four different iterations. At top left is the tree after three iterations, with lower bound 12.6 and upper bound 21. At top right is the tree after five iterations, with lower bound 13.1 and upper bound 20. Note that there is a pruned node, shown with a dotted line. It had a lower bound above 20, the global upper bound at that point, and so could be eliminated from the tree. At bottom left, after 10 iterations, the lower bound is 13.9 and upper bound 19. Finally, the plot at bottom right shows the tree at the end of the algorithm, with lower bound 18.04 and upper bound 19.

Figure 4 shows the evolution of the global lower and upper bounds. We can see, for example, that a point with globally minimum cardinality is obtained after only 8 iterations; however, it takes around 100 more iterations to *guarantee* that this point has minimum cardinality.

Figure 5 shows the portion of non-eliminated or non-pruned sparsity patterns, versus iteration number. Each pruned node, at level i, corresponds to 2^{n+1-i} possible values of the Boolean variable z; each active (non-pruned) node at level i corresponds to 2^{n+1-i} possible values of the Boolean variable. Near termination, the fraction of non-pruned Boolean values is around 1%. This means we have eliminated around 99% of the possible Boolean values. However, the total number of Boolean values is $2^{30} \approx 10^9$, so 1% still represents around 10^7 possible values for z. This means that, right before the algorithm terminates, there are still

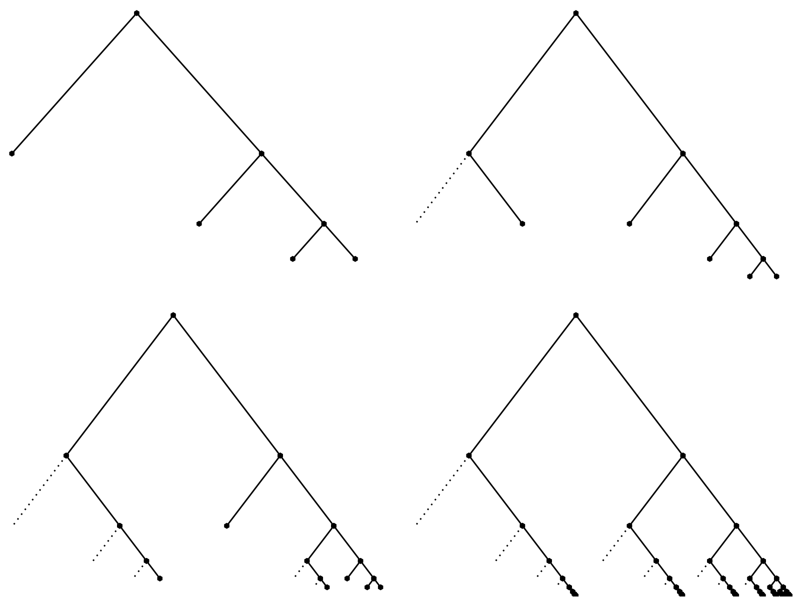

Figure 3: Tree with 30 variable problem, after 3 iterations (top left), 5 iterations (top right), 10 iterations (bottom left), and 124 iterations (bottom right).

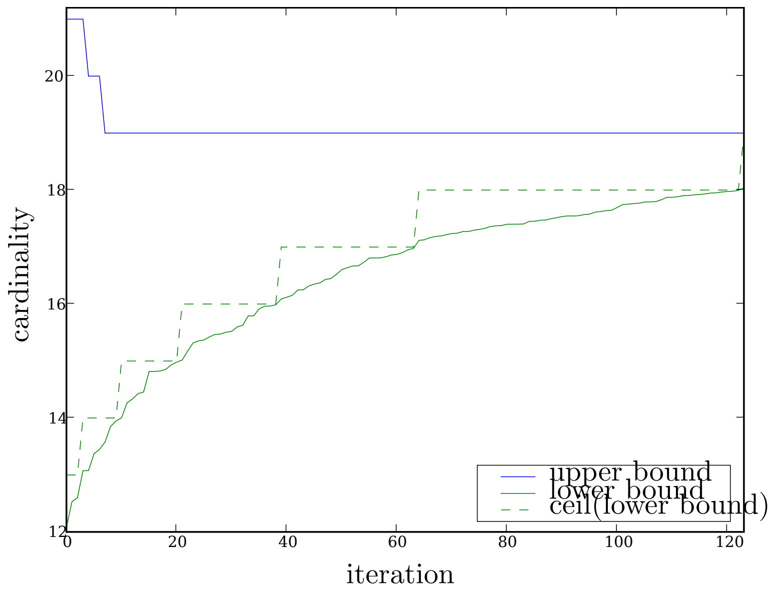

Figure 4: Bounds with 30 variable cardinality example.

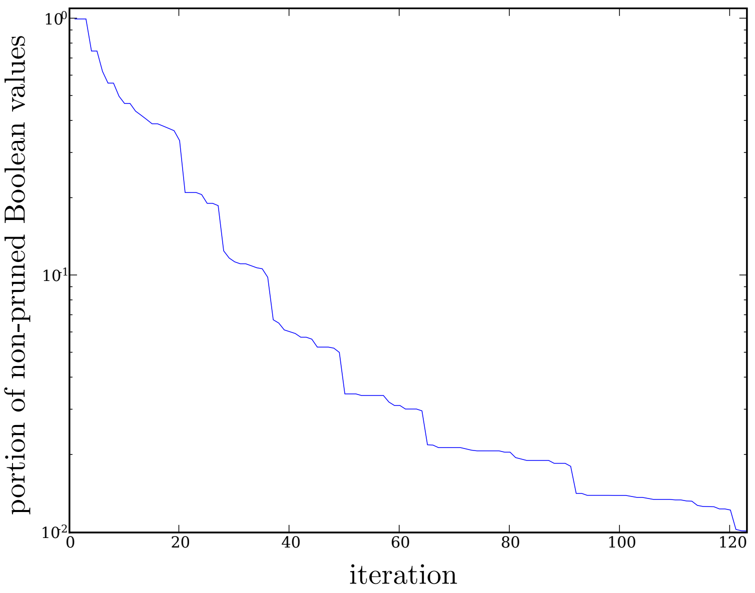

Figure 5: Portion of solutions remaining possible (not eliminated), 30 variables.

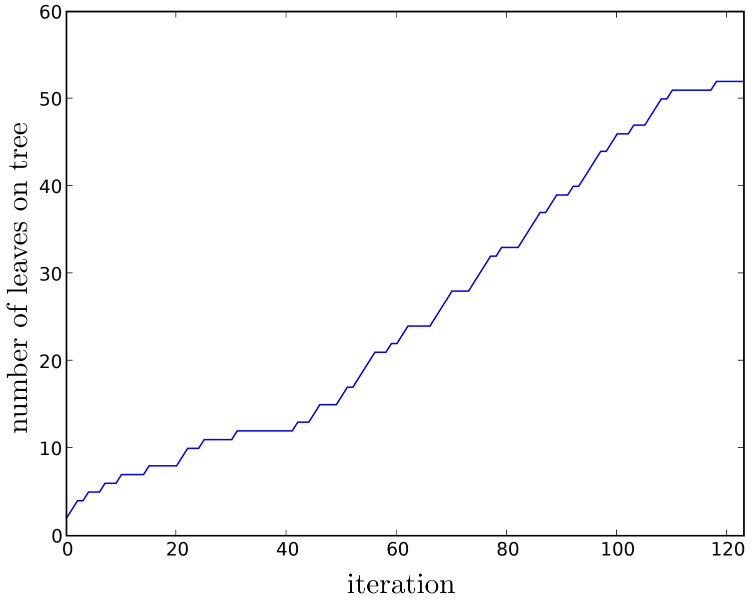

Figure 6: Number of leaves on tree, 30 variables.

10^7 possible values of z 'in play'.

Figure 6 shows the total number of active (non-pruned) leaves in the tree, which is the size of the partition, versus iteration number. After k iterations, we can have at most k leaves (which occurs only if no pruming occurs). So we can see that in this example, we have typically pruned around half of the leaves.

For this example, direct enumeration of all sparsity patterns is impractical (or at least, very slow compared to branch and bound), since it would require the solution of $2^{30} \approx 10^9$ LPs. We must count ourselves lucky to have solved the problem with (provable) global optimality by solving only a few hundred LPs.

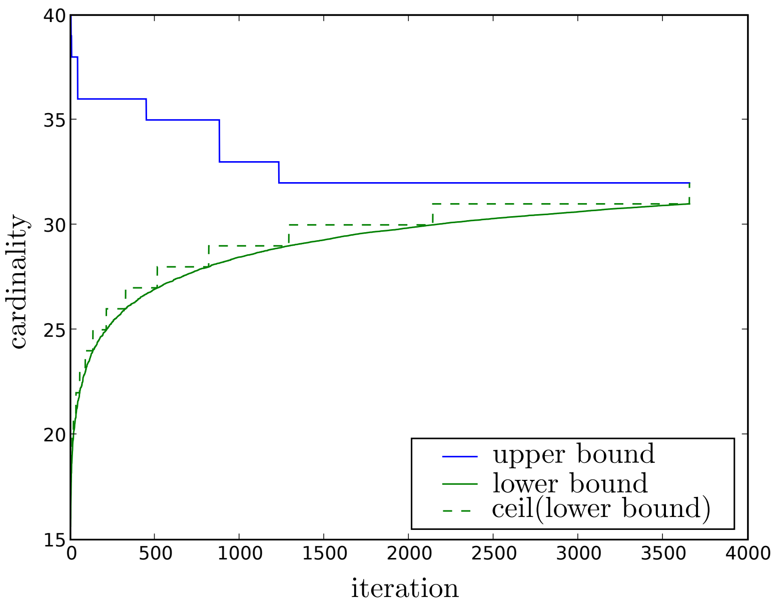

Figure 7: Bounds with 50 variable example.

Larger example

We show the same plots for a larger example with 50 variables. It takes 3665 iterations to find the optimal cardinality of 32. This took approximately 20 minutes on a Linux system with an Intel Core 2 Duo 1.67 GHz and 2 GB of RAM. Solving the problem by brute force would, in comparison, involve, as a minimum, eliminating all sparsity patterns with cardinality 31. In this example, we know that six of the variables cannot possibly be zero, so we need to eliminate $\binom{44}{31} > 10^{10}$ sparsity patterns by solving an LP for each. This is obviously effectively impossible.

This is the same example used in the lecture on ℓ_1-norm methods for cardinality problems. The simple ℓ_1-norm heuristic yields a point with cardinality 44 (by solving one LP); using the iterated ℓ_1-norm heuristic, we obtain a point with cardinality 36 (by solving 4 LPs). The branch and bound method does not produce a better point than this until around 480 iterations (which requires the solution of 1061 LPs), when a point with cardinality 35 is found. It takes around 1200 iterations before an optimal point is found, and another 2500 iterations after that to prove that it is optimal.

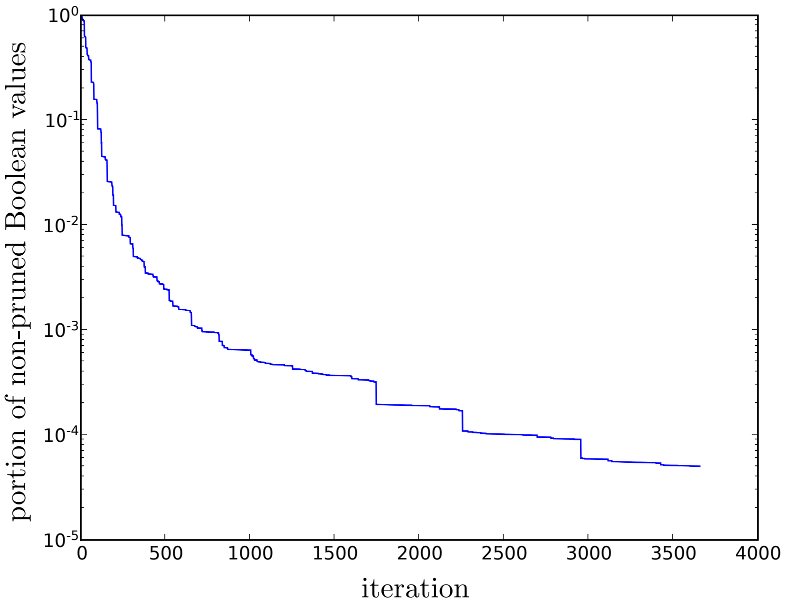

Figure 8: Portion of solutions remaining possible (not eliminated), 50 variables.

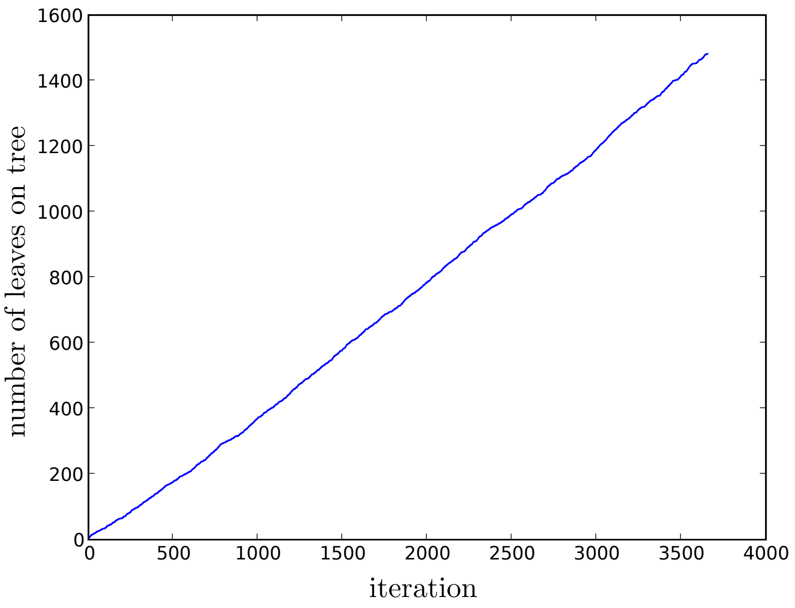

Figure 9: Number of leaves on tree, 50 variables.

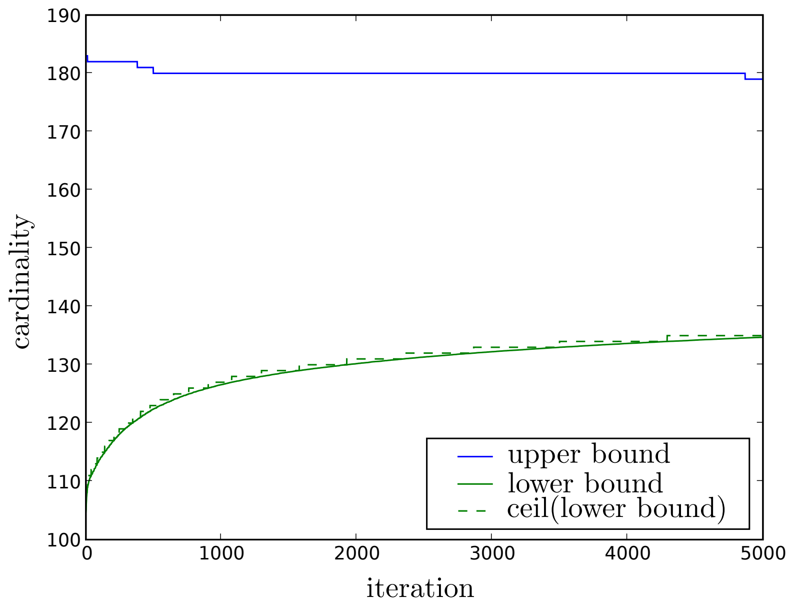

Figure 10: Bounds with 200 variable cardinality example.

An even larger example

In the interest of honesty, we show an example where branch and bound fails (in a practical sense). The problem to be solved here is a randomly generated problem instance of the form (7) with 200 variables. Figures 10, 11, and 12 show partial results from running this algorithm for 50 hours on a Linux system with an Intel Pentium M 1.7 GHz and 1 GB of RAM. After solving around 10000 LPs, we do not know what the optimal cardinality is; we only know that it lies between 135 and 179.

You could, however, report terrific progress to your boss, by pointing out that in 50 hours, you have successfully reduced the total number of possible sparsity patterns (*i.e.*, values of the Boolean variable z) by a factor of 10^{12}.

On the other hand, $2^{200} \approx 1.6 \cdot 10^{60}$, so that still leaves a rather substantial number, $1.6 \cdot 10^{48}$, of sparsity patterns in play. If your boss points this out, you can plead the NP-hard defense.

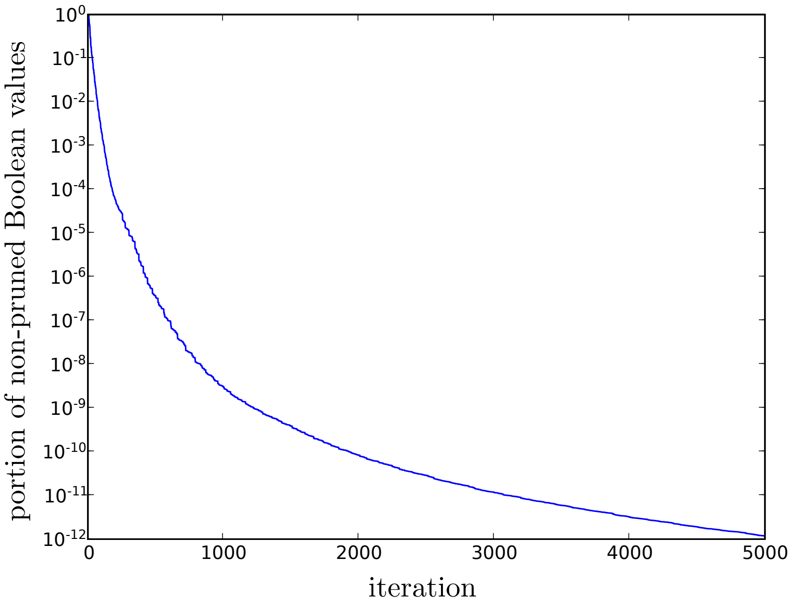

Figure 11: Portion of solutions remaining possible (not eliminated), 200 variables.

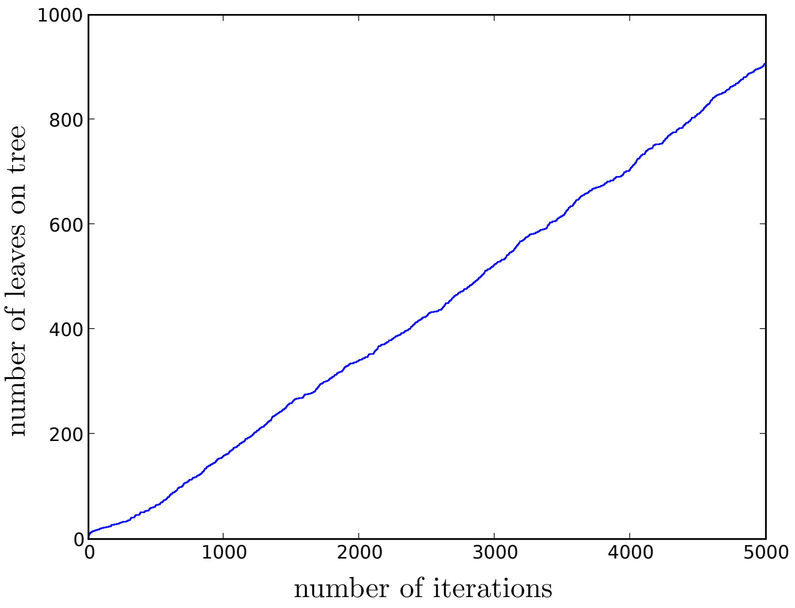

Figure 12: Number of leaves on tree, 200 variables.

Acknowledgments

Arpita Ghosh and Alessandro Magnani helped with an earlier version of this set of notes.