Jose Enrique R. Lopez 16 June 2021

CMSC 142 X-3L Research Project

**Runtime Analysis of Branch-and Bound Solution for the Subset Sum Problem**

The brute-force solution for the subset sum problem is O(2n). The Power Set Law guarantees that the cardinality of all candidate subsets is 2n. Hence, accounting for all possibilities would indeed take exponential time. In other words, if one were to draw a state-space tree of the elements, one would generate 2n nodes.

The branch-and-bound solution improves upon the brute-force solution by excluding elements that no longer need to be pursued, backtracking as necessary. In particular, we explore elements (“branch”) only when the following conditions are satisfied:

1) adding a candidate element would produce a sum less than the target, and

2) there are still elements remaining for the current branch.

Otherwise, we no longer pursue the candidate element and backtrack (“bound”).

More formally, given set S = {S0, S1, …, Si}, target k, current index p, and Boolean set X mapping to S (1 if Si is included, 0 otherwise), we assign two bounding functions for the brute-force algorithm, as follows:

Given that both functions are met, the algorithm keeps adding candidate elements. Otherwise, the algorithm bounds, as reflected in the following if clause from subsetSum.c:

**if (obtained\_weight + option[i][nopts[i]] >= k || nopts[i] ==0){**

**obtained\_weight-=option[i-1][nopts[i-1]];**

**nopts[i-1]--;**

**i--;**

**}**

In addition, this particular implementation also sorts the array before exploration, vastly reducing the number of instances of backtracking. We use qsort from C’s standard library.

We test the branch-and-bound algorithm on the set {1, 2, 3, …, N} with target N+1 to approach worst-case behavior. In this case, almost half of the search tree will be unexplored. For example, for N = 6, the unshaded elements are unexplored:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 6 | 6 | 6 | 6 | 6 | 6 |
| 5 | 5 | 5 | 5 | 5 |  |
| 4 | 4 | 4 | 4 |  |  |
| 3 | 3 | 3 |  |  |  |
| 2 | 2 |  |  |  |  |
| 1 |  |  |  |  |  |

The following graph shows the running time (in secs) of the algorithm given an input size N 250.

The exponential growth of the running time is unmistakable. Though we have reduced our search tree by half, the running time still approaches O (2n). This is because the search tree is reduced only by a constant factor c In other words, O () = O (2n).

However, if we turn this into a decision problem whereby only one solution is required, the running time is somewhat different. In the best case, the subset of the first two elements is immediately selected as a viable solution, yielding O (1) best-case running time. In the worst case, the last two elements are selected, which is identical to searching for all solutions in the reduced search tree, yielding O (2n) worst-case running time.

Other factors may affect the running time of this algorithm. For example, if we set the target to be unreachable, the algorithm behaves in the worst-case since one traverses all the elements of the reduced search tree. Also, if we insist on using an unsorted array, the number of backtracks may increase and in the worst-case, we may result to the same number of branches as the original search tree.

The branch-and-bound solution has two important differences with the dynamic programming solution. Again, if we set the target as unreachable, the two solutions differ in terms of running time. In the former, this is equivalent to searching for all solutions, yielding O (2n) running time. In the latter, one just has to check if the bottom right corner square is 0 and confirm that no solutions exist, yielding O(1) running time. Another difference is that the branch-and-bound solution is able to produce all possible solutions while the dynamic programming solution treats the problem only as a decision problem, and only one solution is generated.

Hence, producing all valid subsets given a target k remains intractable. However, the branch-and-bound solution improves somewhat upon the brute-force solution by a constant factor, though it still runs in O(2n).