CS4800 Programming Assignment

Brian Desnoyers

November 24, 2014

The problem is to maximize the influence value of n employees while only directly influencing k employees.

1 Mathematical Formulation of the Problem

Given a sequence of employees $\{e_1, e_2, e_3, ..., e_n\}$ with bosses $\{b_1, b_2, b_3, ..., b_n\}$ and values $\{v_1, v_2, v_3, ..., v_n\}$, find the benefit of the optimum set of k employees S that maximizes the total benefit where $\text{benefit}_{e_n} = values_n + values_{bosses(e_n) \notin B}$ where $B = S + bosses(S)$.

2 English Description of the Algorithm

At each step in this greedy algorithm, the optimal choice will be to choose the leaf (employee with no subordinates) whose total benefit (benefit of the employee and its bosses who are not already influenced) is the greatest. By doing this k times, the optimal solution will be the sum of the total benefits of each employee added.

3 Pseudo Code Description of the Algorithm

```
def algorithm:
     set solution_value = 0

     calculate the benefit for each leaf

     create a maximum priority queue leaf_queue
     for each employee
          if the employee is a leaf
               add the employee to leaf_queue
               add the employee to its bosses list of employees

     i = k
     while (i > 0)
```

```
      optimal_emp = max-valued employee from leaf_queue
      solution_value += value of optimal_emp
      set that optimal_emp is in solution
      fix_parents(optimal_emp)
      i -= 1

def fix_parents(emp)
     set that emp is in solution
     if emp has a boss and emp's boss is in the solution
          for each leaf in the leaves under emp
               store an operation to subtract the value of emp's value from leaf's benefit
               fix_parents(emp's boss)
     otherwise
          for each employee in the leaves under emp
               store an operation to subtract the value of emp's value from leaf's benefit
               perform any stored operations on this leaf
               heapify down in the priority queue from the location of this leaf
```

4 Proof of Correctness of the Algorithm

Assume the employee *emp* with the greatest benefit at a greedy step is part of an optimal solution S.
Thus, the greedy step is in the optimal solution.

Assume *emp* is part of S.
Then, there exists some employee *emp′* that is part of the optimal solution, but whose benefit $<$ the benefit of *emp*.
Therefore, replacing *emp′* with *emp* would yield a solution with higher benefit than S. This is a contradiction.
Thus, the greedy step is in the optimal solution.

5 Run-time Analysis

let n = the total number of employees
let k = the number of employees to pick
let l = the number of leaves in the tree
let d = the max depth of the tree

For initialization Creating an employee object for each employee and appending that employee to a list of employees takes $O(n)$ time.

Setting up priority queue structure Iterating through the list of employees, inserting the leaves into a priority queue, and adding each leaf to the leaf lists of its bosses takes $O(n(lg(l))$ time.

For each iteration in $1...k$

Finding the optimal employee and updating solution value Finding an optimal employee and fixing the solution value takes $O(1)$ time.

Setting up the solution to find the next optimal employee Setting up the solution to find the next optimal employee by calling the fix_parents method takes $O(dl + llg(l))$ time because fix_parents is called on approximately the height of the tree d and at each level performs an operation on each leaf l in addition to a final update to the priority queue for each affected leaf which takes $llog(l)$ time.

Therefore, overall, the algorithm can run in $O(n(lg(l) + k(dl + llg(l)))$ time. If it is known that the height of the tree is low enough such that the $k(llg(l))$ term dominates, this can be simplified to $O(k(llg(l))$.