CSE 6140 / CX 4140 Assignment 4
due Nov 13, 2020 at 11:59pm on Canvas

1 Local Search and Backtracking [20 pts]

The 3-COLORING problem is as follows: Given a graph G, can you find an assignment of colors to nodes using at most 3 colors, such that no two adjacent nodes have the same color.
3-COLORING is an NP-complete decision problem and hence we will use local search (LS) and backtracking. You will design a local search algorithm to try to find a feasible solution, or at least one for which the number of adjacent nodes with same color is minimized.

(a) (5 pts) Local Search: How will a candidate solution be represented and what will be the evaluation metric for a candidate solution (3 pts)?
Describe one strategy for finding an initial solution (2 pts).

> **Solution:** A candidate solution can be represented as a vector x of dimension n where n equals to the number of vertices in G and $x[i] \in \{0, 1, 2\}$ for $i = 1, 2, ..., n$. Here $x[i] = k$ represents that the ith node is assigned with color k.
>
> We can measure the candidate solution using the number of edges in G having two endpoints with the same number.
>
> We can randomly assign a number in $\{0, 1, 2\}$ to $x[i]$ for each $i = 1, 2, ..., n$.

(b) (5 pts) Local Search: Describe a neighborhood for this search problem – how do we move from one candidate solution to another (3 pts)? What is the size of the neighborhood, i.e. what is the time complexity of finding the best move from the current solution (2 pt)?

> **Solution:** Given a candidate solution x, we can move it to another candidate solution x' where x' differs from x in exactly one location, say j, in the way that $x'[j] = (x[j] + 1)\%3$. The size of the neighborhood will be n.

(c) (10 pts) If we use backtracking to solve the decision problem, what would `Expand()` do for 3-COLORING and how many children will each node have (2 pts)?
What is the worst-case running time of backtracking for 3-COLORING in asymptotic (e.g. Big-O) notation (2 pt)?
What are the possible outcomes of `Check()`? (4pts)
Why would `Check()` prune a search node when solving 3-COLORING (2 pts)?

Solution: The `Check()` step will make choices on three children, which are obtained by considering the three possible colorings the next vertex in the graph may have.

The worst-case running time is $O(3^n)$ since there are 3^n possible ways of coloring n vertices with 3 colors.

Possible outcomes of `Check()` :

1, Find a child to be a solution: If it is better than the best solution, update it to be the best solution, and update its cost to be the best cost.

2, If the child has a cost less than the best cost, we can put the child into the queue for further expansion; otherwise, the child will not be considered further, meaning that we prune this child search node.

`Check()` prunes a search node when it finds that the search node will not give us a better solution. In this way, we can avoid wasting time on unpromising search nodes.

2 Optimizing Amazon's Operations (16 pts)

Amazon is considering building a set of warehouses in a new market. There are n possible locations at which a warehouse can be built, and the cost of building a warehouse at location $i \in \{1, \ldots, n\}$ is $f_i \in \mathbb{R}_{>0}$. To serve this new market efficiently, Amazon would like the warehouses to be close to the cities. In particular, there are m cities, and the distance between city $j \in \{1, \ldots, m\}$ and warehouse $i \in \{1, \ldots, n\}$ is $d(i, j) \in \mathbb{R}_{\geq 0}$. Of course, Amazon could build a warehouse at every one of the n possible locations, but that may be too expensive. Instead, the company would like to construct warehouses at a *subset* W of the n locations, $W \subseteq \{1, \ldots, n\}$, such that the sum of the total construction cost and the minimum distances to the cities is minimized. More formally, Amazon would like to find the set W that minimizes:

$$\sum_{i \in W} f_i + \sum_{j \in \{1,\ldots,m\}} \min_{i \in W} d(i, j)$$

1. Devise a branch-and-bound algorithm for this problem. This entails deciding:

   (a) What is a subproblem? (2 pts)

   Solution: Having made decisions on k locations, can we choose more locations from the rest of $n - k$ locations to build warehouses to minimize the sum of the total construction costs and the distances to the cities?

   (b) How do you choose a subproblem to expand? (2 pts)

   Solution: I will choose the subproblem with the smallest lower bound of the total cost to expand.

   (c) How do you expand a subproblem? (2 pts)

Solution:
1, Choose the location i with the smallest construction fee from the locations that have not been considered.

2, Expand on two decision nodes, one chooses location i and the other dose not choose location i.

(d) What is an appropriate lower bound? Argue why this is a valid lower bound. (3 pts)

Solution: An appropriate lower bound is

$$\min_{i\in\{1,2,\ldots,n\}} f_i + \sum_{j\in\{1,\ldots,m\}} \min_{i\in\{1,2,\ldots,n\}} d(i,j).$$

This is a valid lower bound because $\min(a+b) \geq \min a + \min b$. To minimize the first term in the total cost, we just build one warehouse at the location with the lowest construction cost. To minimize the second term in the total cost, we need to build warehouses at all locations so that each city can be assigned to the nearest possible warehouse. These two extreme situations give us the lower bound I provided above.

2. Outline a simple greedy heuristic for the problem, and provide the pseudocode. (4 pts)
Explain why it finds a valid solution (1 pts) and its running time (2 pts).

Solution: We will first randomly choose one location $i_1 \in \{1, 2, ..., n\}$ and let $W = \{i_1\}$. Then we will repeatedly choose the next location which can decrease the total cost the most until we can no longer decrease the total cost anymore by choosing new locations. Namely, we choose a location k with the largest positive

$$de\_cost(k) = -f_k + \sum_{j=1,\ldots,m} \max(0, \min_{i\in W} d(i,j) - d(k,j))$$

and add k to W until we can no longer find a location with a positive de_cost.

```
def Greedy(f_1, f_2, ..., f_n, d):
    W = {1}
    while we have unselected locations with positve de_cost:
        {
        Select the location i with maximum de_cost(i);
        Add i to W;
        }
    return W
```

My greedy algorithm will always give a non-empty set W, which is ensured to be a valid solution. The worst time complexity is $O(mn^2)$ since the while loop might run n times. During each iteration, computing each $de\_cost(i)$ takes m steps and getting the maximum $de\_cost(i)$ takes a run time of $O(n)$. Therefore, the overall running time is in $O(mn^2)$.

3 Approximation for Bin Packing [14 pts]

Suppose that we are given a set of n objects, where the size s_i of the i^{th} object satisfies $0 < s_i < 1$. We wish to pack all the objects into the minimum number of unit-size bins. Each bin can hold any subset of the

objects whose total size does not exceed 1.

This problem is NP-hard. The *first-fit heuristic* takes each object in turn and places it into the first bin that can accommodate it. Let $S = \sum_{i=1}^{n} s_i$.

(a) Argue that the optimal number of bins required is at least $\lceil S \rceil$ (3 pts).

Solution: We prove by contradiction. Suppose the optimal number of bins required is O and $O < \lceil S \rceil$.Then $O < S$, which is impossible since O bins can not hold all objects with total size of $S > O$. Therefore, $O \geq \lceil S \rceil$.

(b) Argue that the first-fit heuristic leaves at most one bin less than half full (4 pts).

Solution: We prove by contradiction. If there are two bins, bin i and bin j left less than half full, and bin i is left less than half full before bin j, then the first-fit heuristic would put all the objects in bin j into bin i since bin i was the first bin that accommodate those objects. In this way, bin j will be either empty if there is no more object left or more than half full if the objects coming later has a size greater than $1/2$ and dose not fit bin i. So, the first-fit heuristic leaves more than one bin less than half full.

(c) Prove that the number of bins used by the first-fit heuristic is never more than $\lceil 2S \rceil$ (4 pts).

Solution: Suppose the first-fit heuristic uses J bins and each bin holds objects of a total size of b_j for $j = 1, 2, ..., J$. Then we have

$$S = \sum_{j=1,...,J} b_j \geq (J-1) \times \frac{1}{2} + \min_{j=1,...,J} b_j > (J-1)/2$$

as from part (b) we know there are at least $J-1$ bins holding objects of a total size greater than $\frac{1}{2}$ and every bin in use should hold at least one object with size greater than 0. Therefore, we know

$$J < 2S + 1 \leq \lceil 2S \rceil + 1,$$

which together with the fact that J has to be an integer leads us to the conclusion that $J \leq \lceil 2S \rceil$.

(d) Prove an approximation ratio of 2 for the first-fit heuristic (3 pts).

Solution: From results in part (b) and part (c) we know that $J \leq \lceil 2S \rceil \leq 2\lceil S \rceil \leq 2O$. So the first-fit heuristic has an approximation ratio of 2.

4 Bonus: Some DP Problem (7 bonus points)

Evelyn hosted this year's Thanksgiving feast at her place. She invited her family and friends, who all brought gifts for her children. Almost like any other siblings, Charlie and Alan were again fighting over who should get what gifts. Now, to resolve the conflict between them, Evelyn went to seek help from her housekeeper Berta. Berta suggested Charlie and Alan to play the following game:
She arranged all the n gifts in a row having values from $v(1), \cdots, v(n)$, where i^{th} gift has value $v(i)$. Assume n is even. Both of the players get their chances to play, in an alternate fashion. In each turn, a player

can either select the first or the last gift, removes it from the row and gains the corresponding value of the gift. Now, as Charlie is always mean to Alan, Evelyn decides that Alan should go first. Give a dynamic programming algorithm to determine the maximum possible value that Alan can definitely accumulate.

(a) (3 pts) Give the subproblem and prove optimal substructure for this problem.

Solution:
Goal: calculate the maximum possible value $f(1,n)$ that Alan can accumulate if he goes first to select the first or the last gift.
Subproblem: calculate the maximum possible value $f(i,j)$ that Alan can accumulate when it is his turn to select a gift and the remaining gifts are of values $v(i),\cdots,v(j)$. There are two options for Alan:

- Option 1: Alan choose the i^{th} gift. Then Charlie is left with gifts $i+1,\cdots,j$ and the maximum value Charlie can get is $f(i+1,j)$. Since the total value is $\sum_{k=i}^{j} v(k)$, the maximum value Alan can get is $\sum_{k=i}^{j} v(k) - f(i+1,j)$.
- Option 2: Alan choose the j^{th} gift. Then Charlie is left with gifts $i,\cdots,j-1$ and the maximum value Charlie can get is $f(i,j-1)$. Since the total value is $\sum_{k=i}^{j} v(k)$, the maximum value Alan can get is $\sum_{k=i}^{j} v(k) - f(i,j-1)$.

So

$$\begin{aligned} f(i,j) &= \max\Big(\sum_{k=i}^{j} v(k) - f(i+1,j), \sum_{k=i}^{j} v(k) - f(i,j-1)\Big) \\ &= \sum_{k=i}^{j} v(k) - \min(f(i+1,j), f(i,j-1)). \end{aligned}$$

Now we prove that $f(1,n)$ computes correctly the maximum value that Allan can get by induction.

- Base case ($n=1$). $f(1,1) = v(1)$. True since there is only one gift.
- Induction hypothesis: assume $f(i,j)$ is optimal for all $i \le j$ with $j - i = n - 1$.
- When there are n gifts, Allan only has two options, selecting the first gift or the last gift. He should choose the option with the maximum value. After Allan make his decision, Charlie is left with $n-1$ gifts, and he is facing the same problem with Allan. So by the induction hypothesis, the maximum value that Charlie can have is $f(2,n)$ if Allan selected the first gift and $f(1,n-1)$ if Allan selected the last gift. So the maximum value that Allan can get is $f(1,n) = \max(\sum_{k=1}^{n} v(k) - f(2,n), \sum_{k=1}^{n} v(k) - f(1,n-1))$.

(b) (4 pts) Write the recurrence (include base cases).

Solution:

$$f(i,j) = \begin{cases} v(i) & \text{if} \quad i = j \\ \sum_{k=i}^{j} v(k) - \min(f(i+1,j), f(i,j-1)) & \text{otherwise} \end{cases}$$