csci-5454-hw9

Jake Krol

November 2024

1 Question 1

1.1 Question 1a

The **base case is when $\mathbf{v = 0}$**, then $\mathbf{Wt(i, v) = 0}$ since no items should be taken and the total weight is zero. At initialization, the remaining columns in the dynamic programming table can be filled with an arbitrarily large number.

For recurrences, if current item's value exceeds the current value column,($v_i \geq v$), then test whether adding i will reduce the weight by comparing its weight to the entry directly above. Also, if $v_i < v$, then try adding v_i and lookup the most optimal weight for the value difference $Wt(i, v - v_i)$.

$$Wt(i, v) = \begin{cases} min(Wt(i-1, v), w_i) & \text{if } v_i \geq v, \\ w_i + Wt(i, v - v_i) & \text{if } v_i < v \end{cases}$$

1.2 Question 1b

The table size grows as a function of $n \cdot V$. Since filling each entry is done in constant time, the runtime is asymptotic to $\mathbf{O(nV)}$. Where, $V \leq n \cdots v_{max}$ indicates the maximum sum value is less than the number of items multiplied by the maximum value.

Algorithm 1 KNAPSACK(n,V,w)

$n :=$ number of items, $V :=$ value array, $w :=$ weight array

```
Wt = FILL([n, LENGTH(V)], ∞) ▷ n x LENGTH(V) table filled with ∞
Wt[:, 0] = 0                 ▷ fill v = 0 column with 0s
for i = 1 upto n do
    for v_j ∈ V do
        if v_i ≥ v_j then
            Wt(i, v) = min(Wt(i − 1, v), w_i)
        end if
        if v_{j−1} + v_i ≥ v_j then   ▷ test if subset reaches value
            Wt(i, v) = Wt(i, v_j − v_i) + w_i
        end if
    end for
end for
return Wt
```

1.3 Question 1c

Start from the bottom right entry of the table and iterate leftward, testing each element to see if $W_{query} \leq Wt(n, v)$. Once the condition $Wt(i, v) \leq W_{query}$ is met, then v and w have been found. Next, travel upward in the v column until the $Wt(i, v)$ is no longer equal to w. Return the row index of the last element in column v which has the same weight as $Wt(n, v)$. This gives the subset of items $S = \{1 \dots i\}$. The idea is that the bottom row has the best possible weight for each value column. The runtime is $\mathbf{\Theta(V + n)}$ since the worst case involves searching all possible values in the bottom row, then scanning the row indices of one column.

1.4 Question 1d

The max value item value decreases by a factor of K once discretized, and this reduces the column space of the table. The solution is located in the bottom right of the table, which is the last entry filled.

$$\hat{v}_{max} = \lfloor \frac{v_{max}}{K} \rfloor$$

$$O(n \cdot \hat{v}_{max}) \text{ or } \mathbf{O(n \cdot \lfloor \frac{v_{max}}{K} \rfloor)}$$

1.5 Question 1e

The max difference between a discretized item and the original value is

$$v_j - \hat{v_j} \leq k$$

The solution subset contains at most n items, so the difference between the two solutions is at most $k \cdot n$

$$\sum_{j \in S^*}^{n} v_j - \sum_{j \in \hat{S}}^{n} \hat{v_j} \leq k \cdot n$$

Substituting the sums for A and OPT shows that A is no worse than $OPT - K$.

$$\begin{aligned} \sum_{j \in S^*}^{n} v_j - \sum_{j \in \hat{S}}^{n} \hat{v_j} &\leq k \cdot n \\ OPT - A &\leq k \cdot n \\ -A &\leq k \cdot n - OPT \\ \mathbf{A} &\geq \mathbf{OPT - K} \end{aligned}$$

1.6 Question 1f

If $K = \frac{\epsilon \cdot v_{max}}{n}$, then the fact that $v_{max} \leq OPT$ can show that $A \geq OPT(1 - \epsilon)$.

$$\begin{aligned} A &\geq OPT - nK \\ A &\geq OPT - n(\frac{\epsilon v_{max}}{n}) \\ A &\geq OPT - \epsilon v_{max} \\ A \geq OPT - \epsilon \cdot v_{max} &\geq OPT - \epsilon \cdot OPT \\ A &\geq OPT(1 - \epsilon) \end{aligned}$$

1.7 Question 1g

Substituting $K = \frac{\epsilon \cdot v_{max}}{n}$ into the discretized upper bound runtime (solution of 1d), this returns $\mathbf{O}(\frac{\mathbf{n^2}}{\epsilon})$. The approximation algorithm is no longer bounded by the maximum value.

$$O(n \cdot \lfloor \frac{v_{max}}{k} \rfloor) = O(n \cdot \frac{n \cdot v_{max}}{\epsilon \cdot v_{max}}) = O(\frac{n^2}{\epsilon})$$

2 Question 2

2.1 Questions 2a

Given the initial partition implies at least one cut, and since iteratively moving misplaced vertices adds at least one cut to the cut-set, **the maximum**

possible iterations of migrating misplaced vertices is $|\mathbf{E}| - \mathbf{1}$. For example, a vertex v with $deg_{V1}(v) > deg_{V2}(v)$ indicates migrating v to $V1$ will increase the cut-set size. The cut-set strictly increases since the edges between v and $neighbors_{V1}(v)$ is greater than the number of edges between v and $neighbors_{V2}(v)$.

2.2 Question 2b

The total number of edges in a graph is E. For the full graph, the sum of the degree of nodes is $\sum_{v \in V} deg(v) = 2E$. Therefore, total number of edges in the graph is half the degree sum: $E = \frac{1}{2} \sum_{v \in V} deg(v)$.

Next, the disjoint sets $V1$ and $V2$ have no misplaced nodes and this guarantees that the degree of each vertex in it's set after the cut is less than or equal to it's degree in the disjoint set.

Assuming a node is correctly placed in $V1$, let $deg(v)$ be the original degree before any cuts, $deg_{V1}(v)$ be the degree in the $V1$ subgraph, $deg_{V2}(v)$ be the degree in $V2$ subgraph.

$$\begin{aligned} deg_{V1}(v) &\leq deg_{V2}(v) \\ deg_{V1}(v) + deg_{V2}(v) &= deg(v) \\ deg_{V1}(v) + deg_{V1}(v) &\leq deg(v) \\ 2 \cdot deg_{V1}(v) &\leq deg(v) \\ deg_{V1}(v) &\leq \frac{1}{2} \cdot deg(v) \end{aligned}$$

Therefore, the degree of any correctly placed node is less than or equal to half its original degree. Letting v' denote each vertex after its correctly placed, then computing the total degree equation shows the number of edges in the subgraphs must be less than or equal to half the original edges. The remaining edges must be in the cut set $\mathbf{C} \geq \frac{|\mathbf{E}|}{\mathbf{2}}$. Below proves that the worst case, $deg(v') = \frac{1}{2} deg(v)$, results in $\frac{|E|}{2}$ edges present in the graph after the cut; $\frac{|E|}{2}$ edges are in the cut set.

$$\begin{aligned} E &= \frac{1}{2}\Big(\sum_{v' \in V1 \cup V2} \frac{1}{2} \cdot deg(v')\Big) \\ \frac{E}{2} &= \frac{1}{2} \sum_{v' \in V1 \cup V2} deg(v') \end{aligned}$$

2.3 Question 2c

C^* is defined as the largest max-cut possible for a graph. The best case is when C^* can cut all edges in the graph and still result in completely disjoint vertex sets: $\mathbf{C}^* = |\mathbf{E}|$. This is an upper bound on C^*.

In question 2b, it was shown that the algorithm's minimum cut set size is $\mathbf{C} \geq \frac{|\mathbf{E}|}{\mathbf{2}}$. This is a lower bound for C.

Substituting $E = C^*$, we get

$$C \geq \frac{E}{2}$$
$$C \geq \frac{C^*}{2}$$