Greedy algorithms

M. Vazirgiannis

Based on the PC notes “Algorithmes Gloutons”
Minimum spanning tree

• Assume a weighted graph $G = (X, E, w)$
• Spanning tree is a tree that connects all graph nodes
• of minimum spanning tree is the tree with the minimal sum of weights of all spanning tree
• Algorithms:
  – Kruskal’s algorithm
  – Prims algorithm
Dynamic Minimum spanning tree *

Q1: Assume the weights on the edges are distinct - Prove there is a unique minimum spanning tree

A proof of uniqueness by contradiction is as follows

- Say we have an algorithm that finds an MST (which we will call $A$) based on the structure of the graph and the order of the edges when ordered by weight. (Such algorithms do exist, see below.)
- Assume MST $A$ is not unique.
- There is another spanning tree with equal weight, say MST $B$.
- Let $e_1$ be an edge that is in $A$ but not in $B$.
- As $B$ is a MST, $\{e_1\} \cup B$ must contain a cycle $C$ **
- Then $B$ should include at least one edge $e_2$ that is not in $A$ and lies on $C$.
- Assume $w(e_1) < w(e_2)$
- Replace $e_2$ with $e_1$ in $B$ yields the spanning tree: $\{e_1\} \cup B - \{e_2\}$ which has a smaller weight compared to $B$.
- Contradiction! ...we assumed $B$ is a MST but it is not.

* The **dynamic MST** problem adresses the update of a previously computed MST after an edge weight change in the original graph or the insertion/deletion of a vertex
Dynamic Minimum spanning tree

• Q2. Assume A the MST of a graph G. Let e edge which is NOT in A. Reduce the weight of e by \(a > 0\). Propose an algorithm that computes the minimum spanning tree in this case

• justify the complexity, do not care to prove the validity of the algorithm.

• Solution:
  – In \(A \cup \{e\}\) let \(C = \{e, e_1, ..., e_k\}\) a cycle
  – If for all \(i\): \(w(e) > w(e_i) + a\) do not change the tree
  – Else the tree becomes \(A \cup \{e\} - \{e_j\}\) where \(e_j: w(e_j) = \max (w(e_1), ..., w(e_k))\)

Complexity:
- if A is a rooted tree cycle finding is of complexity \(O(n)\), \(n\): # of edges and update the structure with a similar cost
Maximum spanning tree

• A path in the maximum spanning tree is the *widest path* in the graph between its two endpoints: among all possible paths, it maximizes the weight of the minimum-weight edge.

• spanning tree with weight greater than or equal to the weight of every other spanning tree. How can this be found?

• multiply the edge weights by -1 and solving the MST problem on the new graph.
Dynamic Minimum spanning tree

• Q4: Assume A the MST of a graph: increase the weight of an edge in the MST by by $\alpha > 0$. Find the new MST $A'$. 

• Solution:
  – The forest $A - \{e\}$ consists of 2 connected components $C_1$ and $C_2$
  – Let $E'$ all edges of G joining its two components: $E'$ contains $e$
  – Let $\{e'\}$ the edge with the minimum weight in $E'$
    • If $\alpha$ is very small : $e' = e$ and $A' = A$
    • Else $A' = A - \{e\} \cup \{e'\}$

Complexity:
- if A is a rooted tree cycle finding is of complexity $O(n)$, $n$: # of edges and update the structure with a similar cost
Horn clauses satisfiability,

• In formal logic, Horn-satisfiability, or HORNSAT, is the problem of deciding whether a given set of propositional Horn clauses is satisfiable.
• A Horn clause is a clause with at most one positive literal, called the head of the clause, and any number of negative literals, forming the body of the clause.
• A Horn formula is a propositional formula formed by conjunction of Horn clauses.
Horn clauses satisfiability

• Positive implications:
  \((x_1 \land x_2 \ldots \land x_n) \Rightarrow y\), where \(x_i \ldots x_n, y\) are true

• Pure Negative ones: \(!x_1 \lor !x_2 \ldots \lor !x_n\) or
  \((x_1 \land x_2 \ldots \land x_n) \Rightarrow false\)

• Is there an assignment of values that satisfies a set of clauses (a formula) i.e. makes them all true...
Horn clauses satisfiability

• **Q6: Prove that the following greedy algorithm is correct:**
  • Assign *false* to all variables
  • While exists an implication
    \( (x_1 \lor x_2 \ldots \lor x_n) \Rightarrow y \text{ not satisfied, set } y = true \) and use this further..
  • If all pure negative clauses are satisfied
    – return the current assignment
  • Else
    – return ‘‘formula is not satisfiable’’
Example

\[(w \land y \land z) \Rightarrow x, \ (x \land z) \Rightarrow w, \ x \Rightarrow y, \ \Rightarrow x, \ (x \land y) \Rightarrow w, \ (\overline{w} \lor \overline{x} \lor \overline{y}), \ (\overline{z}).\]

- Set all variables to false
- notice \(x\) must be true on account of the singleton implication \(\Rightarrow x\).
- Then \(y\) must also be true, because of \(x \Rightarrow y\).
- …continue
Q6 solution

- The algorithm terminates as in each repetition the number of true variables increases.
- The algorithm is correct, if it returns an assignment, this assignment satisfies both the implications and the negative clauses, thus it is a satisfying truth assignment of the input Horn formula.
- If the algorithm finds no satisfying assignment, then there really is none.

This is so because our rule maintains the following invariant:
- If a certain set of variables is set to true, then they must be true in any satisfying assignment.
- Hence, if the truth assignment found after the while loop does not satisfy the negative clauses, there can be no satisfying truth assignment.
Q7

• **Question 7** Explain how to design a linear time (to the number of literals and clauses) algorithm.

• *Hint: associate a graph to the formula and show how we encode this graph.*
Q7 - solution

• For each clause
  – associate a counter indicating the number of literals to the left of the implication
  – assign the value false in the current assignment.

• Put all the clauses without argument in the list of implications to satisfy.

• For each with an implication,

  – if the variable to the right of implication takes already set to true,
    • there's nothing more to do,
  – else
    • sets it to true,

• decrease the counter associated with implications that the variable appears on the left hand side.
• If a counter reaches zero, the associated clause is shifted to the list of clauses to satisfy.
• The algorithm stops when the list of clauses to be treated becomes empty.
• It remains to evaluate the negative clauses, what is done in linear time.
Q7 solution

- Each decreasing counter is associated with a literal,
- the number of decreasing actions is linear in the size of the formula.
- Each clause is processed at most once.
- decreasing needs constant time

- We can create a graph having
  - a vertex for each implication clause C
  - A vertex for each variable.
  - Edges are
    - from x to C if x is left of the implication and
    - From C to y if y is on the right hand side of the implication.

- This graph represents the formula in terms of adjacency and can be constructed in linear time with linear space complexity to the size of the formula.
Set Cover problem

– Formally, the set covering problem is the following:

• Assume a set of $m$ subsets $S=\{S_1, \ldots, S_m\}$ of the set $U=\{1, \ldots, n\}$
• Find a minimal subset $I$ of $S$ such that $\bigcup_{i \in I} S_i = \{1, \ldots, n\}$
• Set covering decision problem: the input is a pair $(U,S)$ and an integer $k$; the question is whether there is a set covering of size $k$ or less (NP-complete)
• set covering optimization problem, the input is a pair $(U,S)$ and the task is to find a set covering that uses the fewest sets (NP-hard)
Greedy algorithms for set cover

• A new furniture store creates a network of home delivery:
  – each delivery person covers a set of cities,
  – she chooses to pick a minimum number while ensuring that all interesting destinations are covered.
  – Consider a greedy algorithm that selects subsets iteratively at each step by adding a $S_i$ among those that cover as many elements not yet covered.
Greedy algorithms for set cover

• Q8 Form an example that is not optimal with regards to set cover.
  • Assume the set: \{1, 2, 3, 4, 5, 6\}
  • S1 = \{1, 2, 3\} S2 = \{2, 3, 4, 5\} S3 = \{4, 5, 6\} S4 = \{6\}
  • Set cover = \{S1 = \{1, 2, 3\}, S3 = \{4, 5, 6\}\}

• Q9: Prove that if the set cover is consists of k subsets then at least one of them has size \geq n/k elements
Greedy algorithms for set cover

Q10. Let \( n_j \) is the number of items not covered after the greedy algorithm has selected the first \( j \) subsets. Prove:
\[
n_j < n_{j-1} (1-1/k) \quad \text{and then} \quad n_j < n e^{(-j/k)} \quad \text{do} \quad j = k.
\]

Solution: The last set insertion causes an increase in coverage
\[
> n_{j-1} /k \quad \text{therefore}
\]
\[
n_{j-1} - n_{j-1}/k > n_j \Rightarrow n_j < n_{j-1} (1-1/k) \ (1)
\]
Also as \( 1-x<e^{-x} \) & \( n_j < n(1-1/k)^j \Rightarrow n_j < n e^{(-j/k)} \)

- Moreover find the # of necessary repetitions of the greedy algorithm to cover all the elements
- are covered when \( j > k \ log n \)
Optimal Caching – an exchange strategy

The problem:
• There is a small amount of data that can be accessed very quickly, and
• a large amount of data that requires more time to access. You must
decide which pieces of data to have close at hand.

Applications:
• Computer systems: CPU Cache, RAM, Hard Drive
• Computer networks: Web Caching, Content Caching
• End-host systems: Local Caching at the browser

Caching:
• the process of storing a small amount of data in a fast memory to
reduce time spent in fetching from slow memory.

Optimal Caching – an exchange strategy

• Effective caching: when you go to access a piece of data, it is already in the cache.
• cache hit: cache hit occurs when data is readily accessible from the cache.
• A cache maintenance algorithm determines what to keep in the cache and what to evict from the cache when new data needs to be brought in.

The problem:
• consider a set U of n data stored in a main memory.
• a faster memory, the cache, that can hold k < n pieces of data at any time.
• assume that the cache initially holds some k items.
• A sequence of data items D = d1; d2; ... dm drawn from U is presented to us (offline caching),
• we must decide at all times which k items to keep in the cache in order to minimize access to main memory
Optimal Caching – an exchange strategy

cache miss: when item $d_i$ is present, we have a hit, otherwise we are required to bring it from main memory and we have a miss.

- The problem: If we have a miss and the cache is full, we need to evict some other piece that is currently in the cache to make room for $d_i$.

Hence, on a particular sequence of memory references, a cache maintenance algorithm determines an eviction schedule, specifying which items should be evicted from the cache at which point in the sequence.

This determines:
- The contents of the cache
- The number of misses over time

Our objective:
We want to design an algorithm that determines an eviction schedule that minimizes the number of misses.
Optimal Caching – an exchange strategy

- Example:
  - $k = 2$, initial cache = $ab$,
- requests: $a$, $b$, $c$, $b$, $c$, $a$, $a$, $b$.
- Optimal eviction schedule: 2 cache misses
Cache Policy and an argument exchange

- Q12: Assume the sequence of requests: \textit{abdabecbcabaddbc} and the 3-elements capacity cache contains \{a,b,c\} at the starting point. What is the minimum number of transfers from memory to cache?

\textbf{Solution}

<table>
<thead>
<tr>
<th>M</th>
<th>a</th>
<th>b</th>
<th>d</th>
<th>a</th>
<th>b</th>
<th>e</th>
<th>c</th>
<th>b</th>
<th>c</th>
<th>a</th>
<th>b</th>
<th>a</th>
<th>d</th>
<th>b</th>
<th>a</th>
<th>c</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C</td>
<td>a</td>
<td>a</td>
<td>a</td>
<td>a</td>
<td>a</td>
<td>a</td>
<td>a</td>
<td>a</td>
<td>a</td>
<td>a</td>
<td>a</td>
<td>a</td>
<td>a</td>
<td>a</td>
<td>a</td>
<td>a</td>
</tr>
<tr>
<td></td>
<td>b</td>
<td>b</td>
<td>b</td>
<td>b</td>
<td>b</td>
<td>b</td>
<td>b</td>
<td>b</td>
<td>b</td>
<td>b</td>
<td>b</td>
<td>b</td>
<td>b</td>
<td>b</td>
<td>b</td>
<td>b</td>
</tr>
<tr>
<td></td>
<td>c</td>
<td>c</td>
<td>d</td>
<td>d</td>
<td>d</td>
<td>e</td>
<td>c</td>
<td>c</td>
<td>c</td>
<td>c</td>
<td>c</td>
<td>c</td>
<td>c</td>
<td>d</td>
<td>d</td>
<td>d</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>E</td>
<td>c</td>
<td>d</td>
<td>e</td>
<td>c</td>
<td>d</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

|