# Approximation Algorithms

Subhash Suri

June 5, 2018

## 1 Figure of Merit: Performance Ratio

- Suppose we are working on an optimization problem in which each potential solution has a positive cost, and we want to find one of near-optimal cost. When the objective is to minimize the cost, the near-optimal is always going to be more than the optimal. When the objective is to maximize profit, the near-optimal is less that the optimal.
- The standard measure (figure of merit) in Theory of Algorithms to measure the quality of approximation is the ratio:

$$\rho(n) \quad = \quad \max\left(\frac{cost(A)}{cost(OPT)}\right),$$

where the max is over all input instances of size $n$, and the objective function is minimizing. If the objective is a maximization, then we choose

$$\rho(n) \quad = \quad \max\left(\frac{\text{profit}(OPT)}{\text{profit}(A)}\right)$$

## 2 Approximating the Vertex Cover

- Given a graph $G = (V, E)$, find a vertex cover $C \subseteq V$ of minimize size; that is, for each edge $(u, v) \in E$, either $u$ or $v$ is in $C$.
  - $cost(A)$ = size of VC found by our algorithm $A$;
  - $cost(OPT)$ = optimal VC size
  - $\rho(n)$ = max (worst-case) value of $cost(A)/cost(OPT)$ over all $n$ node graphs

- This problem seems well-suited for greedy strategies. Perhaps the most obvious and natural greedy scheme is the following:
  - repeatedly choose the vertex that covers most edges until no edges left uncovered.
- The number of edges covered by a vertex $v$ is its degree $d(v)$. So, we repeatedly choose the max-deg vertex, delete all of its incident edges, and repeat, until no edges left.
- An example graph.
- When does it not find the optimal? How poorly can this algorithm perform in worst case? Is the worst-case ratio bounded (doesn't grow with $n$)?
- Unfortunately not!
- Counter-example is bipartite graph: $G_n = (L + R, E)$, where $L$ is a set of $n$ vertices $1, 2, \ldots, n$.
- Then, for each $i = 2, 3, \ldots, n$, we add another set of vertices $R_i$ where $|R_i| = \lfloor n/i \rfloor$, and each vertex of $R_i$ is then joined to $i$ distinct vertices of $L$. Thus, each vertex of $R_i$ has degree $i$.
- $R_n$ consists of 1 vertex, connected to everyone in $L$. $R_{i-1}$ also consists of a single vertex, connected to $1, 2, \ldots, n-1$. $R_2$ has $n/2$ vertices, each of degree 2.
- If we run the greedy algorithm on this example, we will first choose $R_n$, then $R_{n-1}$, and so on until we choose all the vertices of $R$. How many vertices are in R?

$$n/2 + n/3 + n/4 + \ldots 1 \quad = \quad \Omega(n \log n).$$

- So, $cost(A) = \Theta(n \log n)$.
- However, OPT could have just chosen all the vertices of $L$, of which there are only $n$.
- So, the ratio is at least $\Omega(\log n)$!
- Now, instead consider another simple greedy scheme:

```
– while $E$ not empty,
    1. pick an arbitrary edge $e = (u, v)$
    2. add both $u$ and $v$ to $C$
    3. delete all edges from $E$ incident to $u$ or $v$
```

- Example.
- **Theorem:** The second greedy Vertex Cover algorithm has size at most twice the optimal.
- **Analysis**. Let $A$ be the set of edges picked by the greedy. Since no two edges in $A$ can share a vertex, each of them requires a separate vertex in OPT to cover. So, $OPT \geq |A|$. On the other hand, our greedy cover has size $|C| \leq 2|A|$. QED.
- Interestingly, the more natural greedy strategy of repeatedly picking the vertex of max degree can only achieve an approximation ratio $\log n$.

## 3 Approximating the Traveling Salesman Tour

- Input: $G = (V, E)$, a complete graph, where each edge $e$ has a positive cost $w(e)$.
- Output: The minimum cost tour visiting all the vertices of $G$ exactly once.
- **General Impossibility Theorem.** There is no polynomial-time algorithm to approximate TSP to any factor unless $NP = P$.
- **Proof.**
  - Suppose there is an algorithm that guarantees factor $X$ approximation for TSP in polynomial time. We prove its impossibility by showing that such an algorithm can solve the HAMILTON cycle problem in polynomial time.
  - Given an instane $G = (V, E)$ of the HAM problem, construct a TSP instance $G'$ as follows.
  - Set $w(e) = 1$ if $e \in E$, and $w(e) = (nX + 1)$ if $e \notin E$.
  - Now ask if $G'$ contains a TSP of cost $\leq n$.
  - If $G$ contains a HAM cycle, then the corresponding TSP has cost $n$, using the HAM cycle edges, each of cost one.
  - The approximation algorithm $A$ must find a tour with cost $\leq nX$.
  - Since each edge that is not in $G$ has cost $> nX$, it cannot use that edge. So, the only tours that lead to acceptable approximation are the HAM cycles in $G$.
  - Conversely, if TSP returns a tour that costs more than $nX$, it must mean that $G$ does not contain a HAM cycle.

### 3.1 TSP with Triangle Inequality

- The good news about TSP is that if the edge costs satisfy triangle inequality, that is, $w(a,b) \leq w(a,c) + w(c,b)$, for any $a, b, c$, then we can approx the tour quite well.
- Factor 2 approximation using MST.

## 4 Load balancing: minimizing MakeSpan

- **Problem:** Given a set of $m$ machines $M_1, \ldots, M_m$, and a set of $n$ jobs, where job $j$ needs $p_j$ time for processing, the goal is to schedule the jobs on these machines so all the jobs are finished as soon as possible.
- That is, find the minimum time (called MakeSpan) $T$ by which all jobs can be executed collectively.
- Let $A_i$ be the set of jobs assigned to machine $M_i$. Then, $M_i$ needs time

  $$T_i \quad = \quad \sum_{j \in A_i} p_j$$

  this is called the load on machine $M_i$. We wish to minimize

  $$T \quad = \quad max_i\{T_i\},$$

  which is called the makespan.
- The decision problem is $NP$-complete: it's in $NP$ because we can decide whether the makespan is $T$ or not. For completeness, we can reduce subset sum to it (scheduling on two machines).
- We show that the following simple greedy method gives good approximation. The algorithm makes one pass over the jobs in any order, and *assigns the next job $j$ to the machine with the lightest current load.*
- for $j = 1, \ldots, n$

  1. Let $M_i$ be the machine with the minimum $T_i$ among the machines
  2. Assign $j$ to $M_i$
  3. $A_i \ = \ A_i \cup \{j\}$
  4. $T_i \ = \ T_i + p_j$

- For instance, given 6 jobs, with sizes $2, 3, 4, 6, 2, 3$, and 3 machines, the algorithm gives

$$(2, 6), (3, 2), (4, 3),$$

for the makespan of 8. The optimal makespan is 7: $(3, 4), (6), (2, 2, 3)$.

- **Analysis:** Let $T$ be the makespan produced by the algorithm, and $T^*$ the optimal.
- Note that, unfortunately, we do not know the value of $T^*$. Nevertheless, we can use a lower bound for $T^*$ to achieve an approximation guaranteee.
- A simple lower bound comes from the following trivial observation: since there is a total of $\sum_j p_j$ amount of work to be done by the $m$ machines collectively, the makespan cannot be less than the avg load. Thus,

$$T^* \geq \frac{1}{m} \sum_j p_j \tag{1}$$

- Unfortunately, this lower bound itself can be too weak: for instance, if we have one extremely long job, then the best we can do is to put it by itself on a single machine, but still the makespan has to be as long as this job. This greedy solution is optimal, but the lower bound may be way off, if the remaining jobs are all very small. But this suggests another lower bound:

$$T^* \geq max_j\{p_j\} \tag{2}$$

- **Theorem:** The greedy makespan produces an assignment with makespan $T \leq 2T^*$.
- **Proof.** Look at the machine that attains the max load (makespan). Let this be $M_i$.
- Let $j$ be the last job assigned to $M_i$.
- Key Observation: When $j$ was assigned, $M_i$ was the machine with the least load! Its load just before assignment of $j$ was $T_i - p_j$.
- Since this was the lightest load, *every* other machine has load at least $T_i - p_j$. Thus, adding up all these load, we have

$$\sum_{k=1}^{n} p_k \geq m \times (T_i - p_j)$$

equivalently,

$$T_i - p_j \leq \frac{1}{m} \sum_{k=1}^{n} p_k$$

- But the sum on the right is just the sum of all the jobs, since each job is assigned to some machine. The quantity on the RHS is just the lower bound from (1), and thus

$$T_i - p_j \;\le\; T^*$$

- Now we account for the last job. Here we simply use the inequality (2), which says that $p_j <= T^*$.

  Thus, $T_i \;=\; (T_i - p_j) + p_j \quad \le 2T^*$.

  QED.

- It is easy to construct examples where this bound is tight. There is a better approximation: if we first sort the jobs in the decreasing order of lengths, and assign them using the greedy strategy, then one can show the approximation factor 3/2.

## 5 Set Cover

- There is set $U$ of $n$ elements, and a list $S_1, \ldots, S_m$ of $m$ subsets of $U$. A Set Cover is a collection of these sets whose union is $U$. Each set $S_i$ has a cost (or weight) $w_i$, and our goal is to find a *Set Cover of minimum weight.*

- Imagine $U$ is a set of n baseball cards you wish to collect. The market offers bundles $S_1, \ldots, S_m$ that are subsets of $U$, at prices $w_1, \ldots, w_m$. You wish to collect all the cards in $U$ at the minimum possible total cost. This is the set cover problem.

- This is a fundamental NP-complete problem. We will develop a simple greedy algorithm for it, although the analysis is not trivial.

- The first idea for the greedy is to repeatedly choose the largest set in the list. But this may not be good if the next set includes most of the items already covered (procured). So, a better idea may be to choose the next set that covers most items currently not covered!

- Greedy-Set-Cover

```
1. Initialize $R = U$; (set of remaining uncovered items)
2. while $R$ not empty
   (a) Select set $S_i$ that minimizes $\frac{w_i}{|S_i \cap R|}$
   (b) Delete elements of $S_i$ from $R$
3. return selected sets
```

- Picture

## Analysis of the algorithm

- The sets chosen by the algorithm clearly form a Set Cover. The main question is how much larger is the weight of this cover than the optimal $w^*$.
- Like the load balancing problem, we will need a lower bound for the optimal to compare, but unlike that problem, finding a good lower bound is more subtle and non-trivial.
- Let us look at our greedy "heuristic" and investigate the intuitive meaning of the ratio: $w_i/|Si \cap R|$
- We can think of this as the *cost paid* for covering each new item. Suppose we "charge" this cost $c_s$ to each of the items $s$ newly covered. Note that each item is charged a cost *only* once—when it is covered for the first time.
- For bookkeeping only purpose, let's add this line to the Greedy Algorithm when $S_i$ is added, to account for this cost.
- First note that when $S_i$ is added, its weight is distributed even among the newly covered elements. Thus these costs simple account for the weights of the sets in the cover.
- (Lemma 1:) If $C$ is the greedy set cover, then

$$\sum_{S_i \in C} w_i \quad = \quad \sum_{s \in U} c_s$$

- The key to the proof is to ask: fix a particular set $S_k$. *How much cost $c_s$ all the elements of $S_k$ can incur?*
- In other words, compared to $w_k$, how large is $\sum_{s \in S_k} c_s$? We derive an upper bound for any set $S_k$, even those not selected by the greedy.
- (Lemma 2:) For any set $S_k$,

$$\sum_{s \in S_k} c_s \quad \leq \quad w_k \times H(|S_k|),$$

  where $H(n) = 1 + 1/2 + 1/3 + \ldots + 1/n$ is the Harmonic number.
- (Proof.) To simplify the notation, lets assume that the items of $S_k$ are the first $d = |S_k|$ items: $s_1, s_2, \ldots, s_d$.
- Further, also assume that these items are labeled in the order in which they are assigned cost $c$ by the greedy. (This is just renaming of items.)

- Now consider the iteration in which item $s_j$ is coverd by the greedy, for some $j \leq d$. At the start of the iteration, $s_j, s_{j+1}, \ldots, s_d \in R$ (because of our labeling).
- Thus, $|S_k \cap R|$ is at least $(d - j + 1)$. Therefore, the average cost of items covered by $S_k$ is

$$\frac{w_k}{|S_k \cap R|} \quad \leq \quad \frac{w_k}{d - j + 1}$$

- This is not necessarily an equality because several elements $j', j' + 1, \ldots, j, j + 1, \ldots$ may get covered in one step.
- Suppose the set chosen by the Greedy for this iteration is $S_i$, so the average cost of $S_i$ has to be less than the average cost of $S_k$.
- The key observation is that it's the average cost of $S_i$ that get assigned to $s_j$. So, we have

$$c_{s_j} \;=\; w_i/(|S_i \cap R|) \;\leq\; w_k/(|S_k \cap R|) \;\;\leq\;\; w_k/(d - j + 1)$$

- We now add up all these costs for all items of $S_k$:

$$\begin{aligned} \sum_{s \in S_k} c_s &= \sum_{j=1}^{d} c_s j \\ &\leq \sum_{j=1}^{d} w_k/(d - j + 1) \\ &= w_k(1/d + 1/d - 1 + ...1/2 + 1) \\ &= w_k \times H(d). \end{aligned}$$

- Let $d^* = \max |S_i|$, the size of the largest set. Now comes the final part:
- (Theorem:) The greedy set cover has weight at most $H(d^*) \times w^*$.
- Proof. Let $C^*$ be the optimal set cover, so $w^* = \sum_{S_i \in C^*} w_i$.
- For each of these sets, the previous result implies that

$$w_i \geq 1/H(d^*) \times \sum_{s \in S_i} c_s$$

- But these sets form a set cover, so

$$\sum_{S_i \in C^*} \sum_{s \in S_i} c_s \;\geq\; \sum_{s \in U} c_s$$

- So, we now have

$$\begin{aligned}
w^* &= \sum_{S_i \in C^*} w_i \\
&\geq \sum_{S_i \in C^*} (1/H(d^*) \times \sum_{s \in S_i} c_s) \\
&\geq 1/H(d^*) \times \sum_{s \in U} c_s \\
&\geq 1/H(d^*) \sum_{S_i \in C} w_i \\
&\geq 1/H(d^*) w_C
\end{aligned}$$