Chapter 7

Approximation algorithms II

By Sariel Har-Peled, December 17, 2012[1] Version: 0.5

7.1 Max Exact 3SAT

We remind the reader that an instance of 3SAT is a boolean formula, for example $F = (x_1 + x_2 + x_3)(x_4 + \overline{x_1} + x_2)$, and the decision problem is to decide if the formula has a satisfiable assignment. Interestingly, we can turn this into an optimization problem.

Max 3SAT

Instance: A collection of clauses: $C_1, \ldots, C_m$.
Question: Find the assignment to $x_1, ..., x_n$ that satisfies the maximum number of clauses.

Clearly, since 3SAT is NP-Complete it implies that Max 3SAT is NP-Hard. In particular, the formula F becomes the following set of two clauses:

$$x_1 + x_2 + x_3 \qquad \text{and} \qquad x_4 + \overline{x_1} + x_2.$$

Note, that Max 3SAT is a ***maximization problem***.

Definition 7.1.1. Algorithm **Alg** for a maximization problem achieves an approximation factor α if for all inputs, we have:

$$\frac{\mathbf{Alg}(G)}{\mathrm{Opt}(G)} \geq \alpha.$$

In the following, we present a ***randomized algorithm*** – it is allowed to consult with a source of random numbers in making decisions. A key property we need about random variables, is the linearity of expectation property, which is easy to derive directly from the definition of expectation.

Definition 7.1.2 (*Linearity of expectations*.). Given two random variables X, Y (not necessarily independent, we have that $\mathbf{E}\Big[X + Y\Big] = \mathbf{E}\Big[X\Big] + \mathbf{E}\Big[Y\Big]$.

Theorem 7.1.3. *One can achieve (in expectation)* $(7/8)$*-approximation to Max 3SAT in polynomial time. Namely, if the instance has* m *clauses, then the generated assignment satisfies* $(7/8)m$ *clauses in* expectation.

Proof: Let $x_1, \ldots, x_n$ be the n variables used in the given instance. The algorithm works by randomly assigning values to $x_1, \ldots, x_n$, independently, and equal probability, to 0 or 1, for each one of the variables.

Let Y_i be the indicator variables which is 1 if (and only if) the ith clause is satisfied by the generated random assignment and 0 otherwise, for $i = 1, \ldots, m$. Formally, we have

$$Y_i = \begin{cases} 1 & C_i \text{ is satisfied by the generated assignment,} \\ 0 & \text{otherwise.} \end{cases}$$

Now, the number of clauses satisfied by the given assignment is $Y = \sum_{i=1}^{m} Y_i$. We claim that $\mathbf{E}[Y] = (7/8)m$, where m is the number of clauses in the input. Indeed, we have

$$\mathbf{E}\Big[Y\Big] = \mathbf{E}\left[\sum_{i=1}^{m} Y_i\right] = \sum_{i=1}^{m} \mathbf{E}\Big[Y_i\Big]$$

by linearity of expectation. Now, what is the probability that $Y_i = 0$? This is the probability that all three literals appear in the clause C_i are evaluated to FALSE. Since the three literals are instance of three distinct variable, these three events are independent, and as such the probability for this happening is

$$\mathbf{Pr}\Big[Y_i = 0\Big] = \frac{1}{2} * \frac{1}{2} * \frac{1}{2} = \frac{1}{8}.$$

(Another way to see this, is to observe that since C_i has exactly three literals, there is only one possible assignment to the three variables appearing in it, such that the clause evaluates to FALSE. Now, there are eight (8) possible assignments to this clause, and thus the probability of picking a FALSE assignment is 1/8.) Thus,

$$\mathbf{Pr}\Big[Y_i = 1\Big] = 1 - \mathbf{Pr}\Big[Y_i = 0\Big] = \frac{7}{8},$$

and

$$\mathbf{E}\Big[Y_i\Big] = \mathbf{Pr}\Big[Y_i = 0\Big] * 0 + \mathbf{Pr}\Big[Y_i = 1\Big] * 1 = \frac{7}{8}.$$

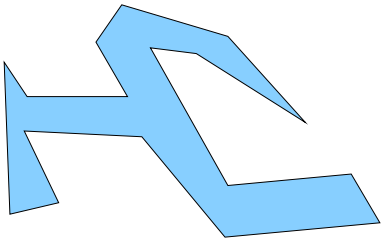
Namely, $\mathbf{E}[\#\text{ of clauses sat}] = \mathbf{E}[Y] = \sum_{i=1}^{m} \mathbf{E}[Y_i] = (7/8)m$. Since the optimal solution satisfies at most m clauses, the claim follows. ∎

Curiously, Theorem 7.1.3 is stronger than what one usually would be able to get for an approximation algorithm. Here, the approximation quality is independent of how well the optimal solution does (the optimal can satisfy at most m clauses, as such we get a $(7/8)$-approximation. Curiouser and curiouser[2], the algorithm does not even look on the input when generating the random assignment.

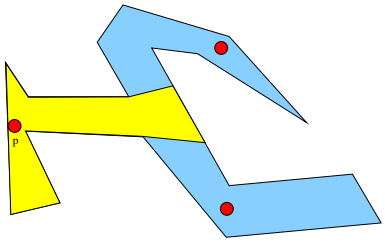
Håstad [Hås01] proved that one can do no better; that is, for any constant $\varepsilon > 0$, one can not approximate 3SAT in polynomial time (unless P = NP) to within a factor of $7/8 + \varepsilon$. It is pretty amazing that a trivial algorithm like the above is essentially optimal.

7.2 Approximation Algorithms for Set Cover

7.2.1 Guarding an Art Gallery



You are given the floor plan of an art gallery, which is a two dimensional simple polygon. You would like to place guards that see the whole polygon. A guard is a point, which can see all points around it, but it can not see through walls. Formally, a point p can ***see*** a point q, if the segment pq is contained inside the polygon. See figure on the right, for an illustration of how the input looks like.



A ***visibility polygon*** at p (depicted as the yellow polygon on the left) is the region inside the polygon that p can see. WE would like to find the *minimal* number of guards needed to guard the given art-gallery? That is, all the points in the art gallery should be visible from at least one guard we place.

The art-gallery problem is a set-cover problem. We have a ground set (the polygon), and family of sets (the set of all visibility polygons), and the target is to find a minimal number of sets covering the whole polygon.

It is known that finding the minimum number of guards needed is NP-Hard. No approximation is currently known. It is also known that a polygon with n corners, can be guarded using $n/3 + 1$ guards. Note, that this problem is harder than the classical set-cover problem because the number of subsets is infinite and the underlining base set is also infinite.

An interesting *open problem* is to find a polynomial time approximation algorithm, such that given P, it computes a set of guards, such that $\#guards \leq \sqrt{n} k_{opt}$, where n is the number of vertices of the input polygon P, and k_{opt} is the number of guards used by the optimal solution.

[2] "Curiouser and curiouser!" Cried Alice (she was so much surprised, that for the moment she quite forgot how to speak good English). – Alice in wonderland, Lewis Carol

7.2.2 Set Cover

The optimization version of Set Cover, is the following:

Set Cover

Instance: $(\mathsf{S}, \mathcal{F})$:
S - a set of n elements
$\mathcal{F}$ - a family of subsets of S, s.t. $\bigcup_{X \in \mathcal{F}} X = \mathsf{S}$.
Question: The set $\mathcal{X} \subseteq \mathcal{F}$ such that $\mathcal{X}$ contains as few sets as possible, and $\mathcal{X}$ covers S. Formally, $\bigcup_{X \in \mathcal{X}} X = \mathsf{S}$.

The set S is sometime called the ***ground set***, and a pair $(\mathsf{S}, \mathcal{F})$ is either called a ***set system*** or a ***hypergraph***. Note, that Set Cover is a minimization problem which is also NP-Hard.

Example 7.2.1. Consider the set $\mathsf{S} = \{1, 2, 3, 4, 5\}$ and the following family of subsets

$$\mathcal{F} = \Big\{\{1,2,3\},\ \{2,5\},\ \{1,4\},\ \{4,5\}\Big\}.$$

Clearly, the smallest cover of S is $\mathcal{X}_{opt} = \Big\{\{1,2,3\},\ \{4,5\}\Big\}$.

The greedy algorithm **GreedySetCover** for this problem is depicted on the right. Here, the algorithm always picks the set in the family that covers the largest number of elements not covered yet. Clearly, the algorithm is polynomial in the input size. Indeed, we are given a set S of n elements, and m subsets. As such, the input size is at least $\Omega(m+n)$ (and at most of size $O(mn)$), and the algorithm takes time polynomial in m and n. Let $\mathcal{X}_{opt} = \{V_1, \ldots, V_k\}$ be the optimal solution.

GreedySetCover$(\mathsf{S}, \mathcal{F})$
  $\mathcal{X} \leftarrow \emptyset$; $T \leftarrow \mathsf{S}$
  while T is not empty **do**
    $U \leftarrow$ set in $\mathcal{F}$ covering largest # of elements in T
    $\mathcal{X} \leftarrow \mathcal{X} \cup \{U\}$
    $T \leftarrow T \setminus U$
  return $\mathcal{X}$.

Let T_i denote the elements not covered in the beginning ith iteration of **GreedySetCover**, where $T_1 = \mathsf{S}$. Let U_i be the set added to the cover in the ith iteration, and $\alpha_i = |U_i \cap T_i|$ be the number of new elements being covered in the ith iteration.

Claim 7.2.2. *We have* $\alpha_1 \geq \alpha_2 \geq \ldots \geq \alpha_k \geq \ldots \geq \alpha_m$.

Proof: If $\alpha_i < \alpha_{i+1}$ then U_{i+1} covers more elements than U_i and we can exchange between them, as we found a set that in the ith iteration covers more elements that the set used by **GreedySetCover**. Namely, in the ith iteration we would use U_{i+1} instead of U_i. This contradicts the greediness of **GreedySetCover** of choosing the set covering the largest number of elements not covered yet. A contradiction. ∎

Claim 7.2.3. *We have $\alpha_i \geq |T_i|/k$. Namely, $|T_{i+1}| \leq (1-1/k)|T_i|$.*

Proof: Consider the optimal solution. It is made out of k sets and it covers S, and as such it covers $T_i \subseteq S$. This implies that one of the subsets in the optimal solution cover at least $1/k$ fraction of the elements of T_i. Finally, the greedy algorithm picks the set that covers the largest number of elements of T_i. Thus, U_i covers at least $\alpha_i \geq |T_i|/k$ elements.

As for the second claim, we have that $|T_{i+1}| = |T_i| - \alpha_i \leq (1-1/k)|T_i|$. ■

Theorem 7.2.4. *The algorithm* **GreedySetCover** *generates a cover of S using at most $O(k \log n)$ sets of $\mathcal{F}$, where k is the size of the cover in the optimal solution.*

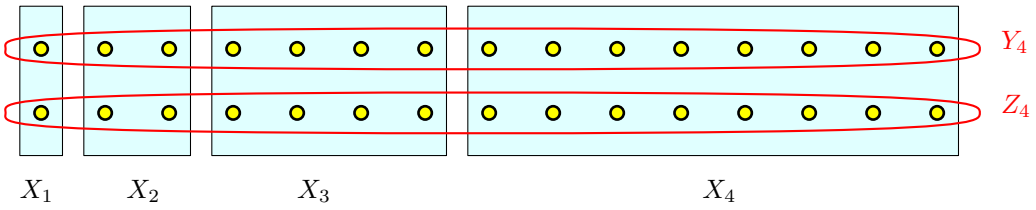
Proof: We have that $|T_i| \leq (1-1/k)|T_{i-1}| \leq (1-1/k)^i |T_0| = (1-1/k)^i n$. In particular, for $M = \lceil 2k \ln n \rceil$ we have

$$|T_M| \leq \left(1 - \frac{1}{k}\right)^M n \leq \exp\left(-\frac{1}{k}M\right) n = \exp\left(-\frac{\lceil 2k \ln n \rceil}{k}\right) n \leq \frac{1}{n} < 1,$$

since $1 - x \leq e^{-x}$, for $x \geq 0$. Namely, $|T_M| = 0$. As such, the algorithm terminates before reaching the Mth iteration, and as such it outputs a cover of size $O(k \log n)$, as claimed. ■

7.2.3 Lower bound

The lower bound example is depicted in the following figure.



We provide a more formal description of this lower bound next, and prove that it shows $\Omega(\log n)$ approximation to **GreedySetCover**.

We want to show here that the greedy algorithm analysis is tight. To this end, consider the set system $\Lambda_i = (\mathsf{S}_i, \mathcal{F}_i)$, where $\mathsf{S}_i = Y_i \cup Z_i$, $Y_i = \{y_1, \ldots, y_{2^i-1}\}$ and $Z_i = \{z_1, \ldots, z_{2^i-1}\}$. The family of sets $\mathcal{F}_i$ contains the following sets

$$X_j = \left\{y_{2^{j-1}}, \ldots, y_{2^j-1},\ z_{2^{j-1}}, \ldots, z_{2^j-1}\right\},$$

for $j = 1, \ldots, i$. Furthermore, $\mathcal{F}_i$ also contains the two special sets Y_i and Z_i. Clearly, minimum set cover for Λ_i is the two sets Y_i and Z_i.

However, sets Y_i and Z_i have size $2^i - 1$. But, the set X_i has size

$$|X_i| = 2\left(2^i - 1 - 2^{i-1} + 1\right) = 2^i,$$

and this is the largest set in Λ_i. As such, the greedy algorithm **GreedySetCover** would pick X_i as first set to its cover. However, once you remove X_i from Λ_i (and from its ground set), you remain with the set system Λ_{i-1}. We conclude that **GreedySetCover** would pick the sets $X_i, X_{i-1}, \ldots, X_1$ to the cover, while the optimal cover is by two sets. We conclude:

Lemma 7.2.5. *Let $n = 2^{i+1} - 2$. There exists an instance of Set Cover of n elements, for which the optimal cover is by two sets, but* **GreedySetCover** *would use $i = \lfloor \lg n \rfloor$ sets for the cover. That is,* **GreedySetCover** *is a $\Theta(\log n)$ approximation to SetCover.*

7.2.4 Just for fun – weighted set cover

Weighted Set Cover

Instance: $(\mathsf{S}, \mathcal{F}, \rho)$:
S: a set of n elements
$\mathcal{F}$: a family of subsets of S, s.t. $\bigcup_{X \in \mathcal{F}} X = \mathsf{S}$.
$\rho(\cdot)$: A price function assigning price to each set in $\mathcal{F}$.
Question: The set $\mathcal{X} \subseteq \mathcal{F}$, such that $\mathcal{X}$ covers S. Formally, $\bigcup_{X \in \mathcal{X}} X = \mathsf{S}$, and $\rho(\mathcal{X}) = \sum_{X \in \mathcal{X}} \rho(X)$ is minimized.

The greedy algorithm in this case, **WGreedySetCover**, repeatedly picks the set that pays the least cover each element it cover. Specifically, if a set $X \in \mathcal{F}$ covered t new elements, then the ***average price*** it pays per element it cover is $\alpha(X) = \rho(X)/t$. **WGreedySetCover** as such, picks the set with the lowest average price. Our purpose here to prove that this greedy algorithm provides $O(\log n)$ approximation.

7.2.4.1 Analysis

Let U_i be the set of elements that are not covered yet in the end of the ith iteration. As such, $U_0 = \mathsf{S}$. At the beginning of the ith iteration, the ***average optimal cost*** is $\alpha_i = \rho(\mathrm{Opt})/n_i$, where Opt is the optimal solution and $n_i = |U_{i-1}|$ is the number of uncovered elements.

Lemma 7.2.6. *We have that:*
(A) $\alpha_1 \leq \alpha_2 \leq \cdots$.
(B) For $i < j$, we have $2\alpha_i \leq \alpha_j$ only if $n_j \leq n_i/2$.

Proof: (A) is hopefully obvious – as the number of elements not covered decreases, the average price to cover the remaining elements using the optimal solution goes up.

(B) $2\alpha_i \leq \alpha_j$ implies that $2\rho(\mathrm{Opt})/n_i \leq \rho(\mathrm{Opt})/n_j$, which implies in turn that $2n_j \leq n_i$. ∎

So, let k be the first iteration such that $n_k \leq n/2$. The basic idea is that total price that **WGreedySetCover** paid during these iterations is at most $2\rho(\mathrm{Opt})$. This immediately implies $O(\log n)$ iteration, since this can happen at most $O(\log n)$ times till the ground set is fully covered.

To this end, we need the following technical lemma.

Lemma 7.2.7. *Let U_{i-1} be the set of elements not yet covered in the beginning of the ith iteration, and let $\alpha_i = \rho(\mathrm{Opt})/n_i$ be the average optimal cost per element. Then, there exists a set X in the optimal solution, with lower average cost; that is, $\rho(X)/|U_{i-1} \cap X| \leq \alpha_i$.*

Proof: Let $X_1, \ldots, X_m$ be the sets used in the optimal solution. Let $s_j = |U_{i-1} \cap X_j|$, for $j = 1, \ldots, m$, be the number of new elements covered by each one of these sets. Similarly, let $\rho_j = \rho(X_j)$, for $j = 1, \ldots, m$. The average cost of the jth set is ρ_j / s_j (it is $+\infty$ if $s_j = 0$). It is easy to verify that

$$\min_{j=1}^{m} \frac{\rho_j}{s_j} \leq \frac{\sum_{j=1}^{m} \rho_j}{\sum_{j=1}^{m} s_j} = \frac{\rho(\mathrm{Opt})}{\sum_{j=1}^{m} s_j} \leq \frac{\rho(\mathrm{Opt})}{|U_{i-1}|} = \alpha_i.$$

The first inequality follows from the fact that if $a/b \leq c/d$ (all positive numbers), then $\frac{a}{b} \leq \frac{a+c}{b+d} \leq \frac{c}{d}$. In particular, for any such numbers $\min\left(\frac{a}{b}, \frac{c}{d}\right) \leq \frac{a+c}{b+d}$, and applying this repeatedly implies this inequality. The second inequality follows as $\sum_{j=1}^{m} s_j \geq |U_{i-1}|$. This implies that the optimal solution must contain a set with an average cost smaller than the average optimal cost. ■

Lemma 7.2.8. *Let k be the first iteration such that $n_k \leq n/2$. The total price of the sets picked in iteration 1 to $k-1$, is at most $2\rho(\mathrm{Opt})$.*

Proof: By Lemma 7.2.7, at each iteration the algorithm picks a set with average cost that is smaller than the optimal average cost (which goes up in each iteration). However, the optimal average cost iterations, 1 to $k-1$, is at most twice the starting cost, since the number of elements not covered is at least half the total number of elements. It follows, that the for each element covered, the greedy algorithm paid at most twice the initial optimal average cost. So, if the number of elements covered by the beginning of the kth iteration is $\beta \geq n/2$, then the total price paid is $2\alpha_1\beta = 2(\rho(\mathrm{Opt})/n)\beta \leq 2\rho(\mathrm{Opt})$, implying the claim. ■

Theorem 7.2.9. **WGreedySetCover** *computes a $O(\log n)$ approximation to the optimal weighted set cover solution.*

Proof: **WGreedySetCover** paid at most twice the optimal solution to cover half the elements, by Lemma 7.2.8. Now, you can repeat the argument on the remaining uncovered elements. Clearly, after $O(\log n)$ such halving steps, all the sets would be covered. In each halving step, **WGreedySetCover** paid at most twice the optimal cost. ■

7.3 Biographical Notes

The Max 3SAT remains hard in the "easier" variant of MAX 2SAT version, where every clause has 2 variables. It is known to be NP-Hard and approximable within 1.0741 [FG95], and is not approximable within 1.0476 [Hås01]. Notice, that the fact that MAX 2SAT is hard to approximate is surprising as 2SAT can be solved in polynomial time (!).

Bibliography

[FG95] U. Feige and M. Goemans. Approximating the value of two power proof systems, with applications to max 2sat and max dicut. In *ISTCS '95: Proceedings of the 3rd Israel Symposium on the Theory of Computing Systems (ISTCS'95)*, page 182, Washington, DC, USA, 1995. IEEE Computer Society.

[Hås01] J. Håstad. Some optimal inapproximability results. *J. Assoc. Comput. Mach.*, 48(4):798–859, July 2001.