Assignment 1 - Homework Exercises on Approximation Algorithms

Pattarawat Chormai - 0978675, Duy Pham - 0980384, Maciej Wikowski

September 21, 2015

A.I-1

We will show that the approximation ratio of the $GreedySchedulingAlgorithm$ is at least $2 - \frac{1}{m}$ by showing an example as follow.

Let's consider this setting:

- 3 machines: M1, M2, M3
- 7 jobs: 1, 1, 1, 2, 1, 1, 2

$GreedySchedulingAlgorithm$ will come up with this scheduling:

- M1: 1 2 2
- M2: 1 1
- M3: 1 1

Thus $ALG = makespan = 1 + 2 + 2 = 5$
We know that:

$$OPT \geq Average_{load} \tag{1}$$

Where $Average_{load} = \frac{1}{m} \sum_{i=1}^{n} j_i = \frac{9}{3} = 3$
Here we can find a solution with $makespan = 3$. That is:

- M1: 1 2
- M2: 1 2

- M3: 1 1 1

Therefore, $OPT = 3$
Thus, the approximation ratio is:

$$\rho = \frac{ALG}{OPT} = \frac{5}{3} \tag{2}$$

According to the theorem, the estimated ratio is:

$$\rho_{estimated} = 2 - \frac{1}{m} = 2 - \frac{1}{3} = \frac{5}{3} \tag{3}$$

From 2 and 3, we have $\rho_{estimated} = \rho$. Therefore, this bound is tight.

A.I-2

From the question, we know that

$$m = 10$$
$$\sum_{j=1}^{n} t_j \geq 1000$$
$$t_j \in [1, 20]\,;\, for\, all\, i \leq j \leq n$$

Let T'_{i^*} denote the load of M_i before t^*_j, last job, is assigned to the machine. Thus T^*_i, which represents makespan of the assignment, equals to

$$T^*_i = T'_{i^*} + t^*_j$$

Because T'_{i^*} is the minimum load among all machines, so that we can derive

$$T'_{i^*} \leq \frac{1}{m} \sum_{i=1}^{m} T'_i = \sum_{j=1}^{j^*} t_j \leq \frac{1}{m} \left[\sum_{j=1}^{n} t_j - t^*_j \right] \leq LB$$

Then we can derive

$$
\begin{aligned}
T_i^* &= T'_{i^*} + t_j^* \\
&\leq \frac{1}{m}\left[\sum_{j=1}^{n} t_j - t_j^*\right] + t_j^* \\
&\leq \frac{1}{m}\sum_{j=1}^{n} t_j + (1 - \frac{1}{m})t_j^* \\
&\leq 100 + (1 - \frac{1}{10})20 \\
&\leq 118
\end{aligned}
$$

According *Algorithm Greedy Scheduling* and the question, we know

$$
max\left(\frac{1}{m}\sum_{j=1}^{n} t_j\,, \max_{1\leq j\leq n}(t_j)\right) \leq LB \leq OPT
$$

$$
\max_{1\leq j\leq n}(t_j) = 20
$$

Then we can derive

$$
max\left(\frac{1}{m}\sum_{j=1}^{n} t_j\,, 20\right) \leq LB
$$

Since $\frac{1}{m}\sum\limits_{j=1}^{n} t_j \geq 1000$, thus

$$
\begin{aligned}
100 &\leq LB \\
&\leq OPT
\end{aligned}
$$

Therefore, approximation-ratio(ρ) equals to

$$
\begin{aligned}
T_i^* &\leq \rho OPT \\
\frac{118}{100} &\leq \rho \\
1.18 &\leq \rho
\end{aligned}
$$

For this particular setting, *Algorithm Greedy Scheduling* is 1.18 approximation algorithm.

A.I-3

(i)

Assume we have the optimal solution, which has n squares. That is:

$$n \leq LB \leq OPT$$

Now we put our set of points into a grid with unit-size cells. Each unit square can overlap at most 4 cells in such a grid, then our optimal solution can be split into at most $4n$ squares. Therefore,

$$ALG \leq 4n = 4.OPT$$

So this algorithm is 4-approximation.

(ii)

We propose the algorithm as follow.

Algorithm 1 Finding minimum row square cover

Require: Set of Points P
Ensure: Min Square Cover min
Operation:

```
set currentCoveringPosition = −1
QuickSortAscending(S)
for all Point p in P do
    if p.x > currentCoveringPosition then
        create square s = (p.x, 1, p.x + 1, 0);
        add s to S
        set currentCoveringPosition = p.x + 1
    end if
    set min = sizeofS
     return min
end for
```

This algorithm is correct because:

- Every point in p will be covered by a square

- There are no intersections between the squares because we traverse in one direction

This algorithm consists of 2 parts: QuickSort and Traversing the Point to create squares. Let t be the run time of this algorithm, $t_{quicksort}$ be the time for quick-sort, and t_{assign} be the time for creating the squares. We have:

$$t = t_{quicksort} + t_{assign} \leq n \log n + n = O(n \log n) \quad (4)$$

Thus the runtime of this algorithm is $O(n \log n)$.

(iii)

The idea of our algorithm is that, we put all the points in to a coordinate system, then we divide the coordinate system into a set of unit rows (i.e. rows with height 1). For each row, we use algorithm 1 to find the minimum size square-cover. The global min-square-cover is the sum of all row-square-cover.

Algorithm 2 Finding global minimum square cover

Input: Set of points P
Output: Min Square Cover *min*
Operation:

```
currentMin = 0;
for all Row r in the space do
    currentMin += FindRowMinSquare()
end for
set min = currentMin
    return min
```

Theorem. FindingGlobalMinimumSC is $2 - approximation$

Proof. We prove this Theorem by induction.

If the optimal solution consists of only 1 square, then $OPT_1 = 1$. After applying our algorithm, the square can be split into at most 2 squares.

This is true because if the algorithm returns more than 2 squares, then there is a row which consists of more than 1 square. It means the margin of our points is larger than 1, then the optimal must have more than 1 squares to fit them. It contradicts with our assumption that $OPT_1 = 1$.

So $ALG_1 \leq 2 = 2OPT_1$

Suppose when $n = k$, the algorithm is true, that is $ALG_k \leq 2OPT_k$

Now we add some addtional points which insert another square into the optimal solution. Now $n = k + 1$.

We have $OPT_{k+1} = OPT_k + 1$

Applying our algorithm, the final result is the sum of the original input ($n = k$) and the new input ($n = 1$). We know that the Theorem holds for both of them. We have:

$$ALG_{k+1} = ALG_k + ALG_1 \leq 2OPT_k + 2 = 2(OPT_k + 1) = 2OPT_{k+1} \quad (5)$$

Thus the Theorem also holds for $n = k + 1$. Therefore, it holds for all values of n.

In conclusion, this algorithm is $2 - approximation$.

□

AII.1

(i)

We prove this statement by contradiction.

- Suppose that $V \setminus C$ is not an independent set of G. Then there exists a pair of vertices (u, v) in $V \setminus C$ which are connected by an edge $e \in E$. Thus, both u and v are not in C. Therefore, C is not the vertex cover of G anymore.

- Suppose C is not the vertex cover of G, then there exists a pair of vertices (u, v) that are connected by an edge $e \in E$ but are not in C. Thus, $u \in (V \setminus C)$ and $u \in (V \setminus C)$. Therefore, $(V \setminus C)$ is not the vertex cover of G anymore.

From the reasoning above, we can state that: C is the vertex cover of G if and only if $V \setminus C$ is an independent set of G.

(ii)

We prove that $ApproxMaxIndependentSet$ is not a 2-approximation algorithm by showing a counter example. That is, consider a complete graph, for example, a graph $G = (V, E)$ where $V = x_1, x_2$ and $E = (x_1, x_2)$.

Applying the $ApproxMinVertexCover(G)$, we get $C = x_1, x_2$ (picking both vertices from the edge).

Now we take the approx max independent set $ALG = V \setminus C = \emptyset$.

The optimal solution now is $OPT = 1$ (picking x_1 or x_2).

The approximation ratio is $\rho = \frac{OPT}{ALG} = \infty \neq 2$.

So the approximation ratio is not 2.

AII.2

(i)

The best possible scenario in the presented case is similar to following:

$$(x_1 \vee x_2 \vee x_3) \wedge (x_1 \vee x_4 \vee x_5) \wedge (x_1 \vee x_6 \vee x_7)$$

It can be seen that in this case x_1 is present in all the clauses of the DNF therefore we can eliminate all the clauses of the equation in the first run. This gives us a following lower bound:

$$LB = 1$$

We deduce then:

$$OPT \geq 1$$

We define approximation ratio ρ as:

$$\rho = \frac{ALG}{OPT}$$

Since we don't know if duplication of elements in a clause is allowed or not we will examine two possible scenarios.

Duplication allowed:

Consider the case when:

$$(x_1 \vee x_2 \vee x_2) \wedge (x_1 \vee x_3 \vee x_3) \wedge (x_1 \vee x_4 \vee x_4) ... \wedge (x_1 \vee x_n \vee x_n)$$

We choose in each iteration elements unique for the clause like $x_2, x_3, x_4 ... x_n$. So we end up having chosen n elements, excluding the one that was common in all the clauses. That gives us:

$$ALG = (n-1)$$
$$\rho = (n-1)$$

Duplication disallowed:

If duplication is not allowed as in the following DNF:

$$(x_1 \vee x_2 \vee x_3) \wedge (x_1 \vee x_2 \vee x_4) \wedge (x_1 \vee x_2 \vee x_5) ... \wedge (x_1 \vee x_2 \vee x_n)$$

We can see that the algorithm chooses $n-2$ resulting in :

$$ALG = (n-2)$$
$$\rho = (n-2)$$

(ii)

For the algorithm to become a 3-approximation algorithm it should be modified so that in each iteration it chooses all three elements in a clause and eliminate all the clauses in DNF that contain any of these three elements.

Proof

Let D^* be a subset of D that only contains clauses that don't share any variables.

The optimal solution to the problem OPT contains at least one variable from each clause therefore :

$$OPT \geq |D^*|$$

In our algorithm after the modification, we select three variables in each clause in D^*, because there are no clauses that share common variables with them. The other clauses which have common variables as any clause in D^* are deleted. Thus:

$$\begin{aligned} ALG &= 3|D^*| \\ &\leq 3OPT \end{aligned}$$

Therefore it is a 3-approximation algorithm.

AII.3

(i)

Suppose a d-hypergraph $G = (V, E)$ which every edge $e \in E$ incidenst to d vertices in V. To formulate 0/1 linear programming, we introduce $X = \{\ x_i,\ x_2,\ ...\ ,\ x_n\ \}$ which x_i represents $v_i \in V$ in a linear programming. If $x_i = 1$, it means we pick v_i to the set of double vertex cover, $C \subset V$, and otherwise $x_i = 0$. For this solution, we want to find a minimum double vertex cover which requires at least 2 vertices from each edge are in C. Then, we can derive a constraint for 0/1 linear programming

.

$$\sum_{v_i \in e} x_i \geq 2 \text{ for all } e \in E\ ;$$

Thus, we then formulate the linear programming.

$$\begin{array}{lll} \text{Minimize} & & \sum_{x=1}^{n} x_i \\ \text{Subject to} & & \\ & \sum_{v_i \in e} x_i \geq 2 & \text{for all } e \in E\ ;\ \text{at least 2 vertices are selected.} \\ & x_i = \{0,1\} & \text{for all } x_i \in X \end{array}$$

(ii)

Because we can not solve 0/1 linear program in polynomial time, what we have to do next is to relax the program to be a normal linear program by replacing {0,1} constraint with $0 \leq x \leq 1$

Thus, the linear program is

$$\begin{aligned} &\text{Minimize} && \sum_{x=1}^{n} x_i \\ &\text{Subject to} && \\ &\quad \sum_{v_i \in e} x_i \geq 2 && \text{for all } e \in E \text{ ; at least 2 vertices are selected.} \\ &\quad 0 \leq x_i \leq 1 && \text{for all } x_i \in X \end{aligned}$$

Let τ denote the rounding threshold such that

$$x_i = \begin{cases} 1, & \text{if} x_i \geq \tau \\ 0, & \text{otherwise} \end{cases}$$

Algorithm 3 Finding double vertex cover

Input: V, E

Output: A minimum double vertex cover

Operation:

Solve the relaxed linear program corresponding to the given problem.

Minimize $\sum_{x=1}^{n} x_i$

Subject to

$\sum_{v_i \in e} x_i \geq$ for all $e \in E$

$0 \leq x_i \leq 1$ for all $x_i \in X$

C $\leftarrow$ { $v_i \in V$: $x_i \geq \tau$ }

return C

The next step is to derive τ such that all constraints are satisfied and the algorithm always return a valid solution. Let denote x^* to be an ideal value of any x_i such that it satisfies all constraints.

$$\begin{aligned}
\sum_{v_i \in e} x_i &\geq 2 \\
\sum_{i=1}^{d} x_i &\geq 2 \\
1 + \sum_{i=1}^{d-1} x^i &\geq 2 \\
(d-1)x^i &\geq 1 \\
x_i &\geq \frac{1}{d-1} \\
\therefore \tau &= \frac{1}{d-1}
\end{aligned}$$

Let denote W denote the value of an optimal to the relaxed linear program and OPT denote the minium number of double vertex cover. Then $OPT \geq W$.

Now we can derive,

$$\begin{aligned}
|C| &= \sum_{v_i \in C} 1 \\
&\leq \sum_{v_i \in C} (d-1)x_i \\
&\leq (d-1) \sum_{v_i \in C} x_i \\
&\leq (d-1)W \\
&\leq (d-1)OPT
\end{aligned}$$

(iii)

Lets take an example of a complete 3-hypergraph, where the optimal double vertex cover is $|V|-1$ to make sure every edge has at least 2 vertices selected. So the result of the 0/1-LP is $|V|-1$.

The relaxed-LP formulation is as follow:

- Miminize $\sum_{i=1}^{n} x_i$
- Subject to: $\sum_{x_j \in e} x_j \geq 2$ for all edge e AND $0 \leq x_i \leq 1$

We run the algorithm by performing that relaxed-LP on the complete 3-hypergraph, and then round the result following the condition $x \geq \frac{1}{2}$.

For the complete 3-hypergraph, the relaxed-LP will return $x_i = \frac{2}{3}$ for all i so that each sum of vertices in an edge is 2.

Then the algorithm will pick all of the vertices because they satisfy the condition. The result is:

$$ALG = \frac{2}{3}|V|$$

The integrality gap, denoted by IG, is:

$$\begin{aligned} IG &= \frac{|V| - 1}{\frac{2|V|}{3}} \\ &= \frac{3}{2} - \frac{3}{2|V|} \end{aligned}$$

AIII-1-i)

We assume total time $T = \sum_{j=1}^{n} t_i$ to be the total time of all the jobs. Since we define large job as having time $t \geq \epsilon T$, we can deduce that if all runing jobs are large jobs, the maximal number of those jobs is equal to :

$$n_{max} = \frac{T}{\epsilon T} = \frac{1}{\epsilon}$$

For each job we take into account there are two possible ways of assigning it to a machine. Therefore the possible ways jobs can be scheduled is:

$$2 * 2 * ... * 2 = 2^{\frac{1}{\epsilon}}$$

Since we don't distuingish between the machines, we remove the duplicates leaving the total number of schedules at :

$$\frac{2^{\frac{1}{\epsilon}}}{2} = 2^{\frac{1}{\epsilon}-1}$$

AIII-1-ii)

To obtain PTAS, we classify the jobs in the schedule according to the description of the problem. We split them up into two sizes :

$$\text{Job is } \begin{cases} \text{Large if } t_j \geq \epsilon T \\ \text{Small if } t_j < \epsilon T \end{cases}$$

We now can achieve the polynomial time by arranging the small jobs into large job sized slices. This gives us the ability to treat all jobs equal when creating a schedule. We can now use a brute-force method to find the most optimal schedule for the problem. Resulting in the following algorithm:

Algorithm 4 Load Balancing PTAS

Input: n-sized array of jobs t_j
Output: A minimal maximum Makespan of the both machines
Operation:
  Split jobs into large and small jobs according to the definition.
  Arrange small jobs into large job sized chunks and create a new array J containing both chunks and large jobs
  Generate an array S of different schedules for the J array of jobs
  Establish $minMakespan := \infty$ and $bestSchedule := null$
  Foreach Schedule in S
    Calculate M_1 and M_2 makespans on the respective machines
    If $max(M_1, M_2) < minMakespan$
      $minMakespan = max(M_1, M_2)$
      $bestSchedule = S$
    Else
      Continue
  return $bestSchedule$

Proof
To prove that the presented algorithm is indeed a PTAS we introduce a variable S_i denoting the total size of small jobs on a machine in an optimal schedule where $1 \leq i \leq 2$, and a variable S being a total processing time for the small jobs in an optimal schedule. Because we now have small jobs arranged into identically sized chunks we can deduce a total number of jobs in a schedule:

$$\lceil S_1/(\epsilon T) \rceil + \lceil S_1/(\epsilon T) \rceil \geq \lfloor S_1/(\epsilon T) + S_2/(\epsilon T) \rfloor = \lfloor S/(\epsilon T) \rfloor$$

We can now deduce that for the optimal solution, by assigning the chunks we can obtain a maximal error of:

$$\lceil S_i/(\epsilon T)\rceil \epsilon T - S_i \leq (S_i/(\epsilon T) + 1)\epsilon T - S_i = \epsilon T = \text{Total Error}$$

From this we figure out that:

$$ALG \leq OPT + \epsilon T \leq (1+\epsilon)OPT$$

Which proves that the algorithm is indeed a PTAS according to the definition.

Running Time
To analyze the running time we conclude that:

- The generation of the possible schedules runs in $O(2^{\frac{1}{\epsilon}})$ because there are $2^{\frac{1}{\epsilon}}$ possible schedules as stated in the first part of the exercise
- The calculation of makespan for all this schedules is similarly $O(2^{\frac{1}{\epsilon}})$.

This states that the algorithm runs in $O(2^{\frac{1}{\epsilon}})$.

AIII.2

(i)

Let d denote the distance between 2 arbitrary vertices corresponding to P and d^* denote the distance after rounding $p_{i,x}, p_{i,y}$ where $p_{i,x}$ and $p_{i,y}$ denote the x- and y-coordinate of $p_i \in P$, by Δ.

$$\begin{aligned} d =& \sqrt{(p_{i,x} - p_{j,x})^2 + (p_{i,y} - p_{j,y})^2} \\ d^* =& \sqrt{(p_{i,x^*} - p_{j,x^*})^2 + (p_{i,y^*} - p_{j,y^*})^2} \end{aligned}$$

We know that the range of p_x^* is

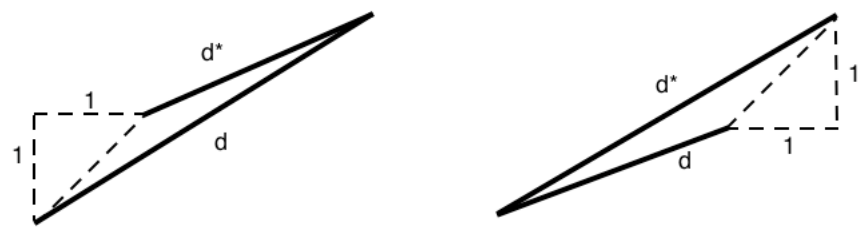
$$\frac{px}{\Delta} \leq p_x^* \leq \frac{px}{\Delta} + 1$$

Then we derive the range of d^*

$$\sqrt{(\frac{p_{i,x}}{\Delta} - (\frac{p_{j,x}}{\Delta} + 1))^2 + (\frac{p_{i,y}}{\Delta} - (\frac{p_{j,y}}{\Delta} + 1)^2} \leq d^* \leq \sqrt{(\frac{p_{i,x}}{\Delta} + 1 - \frac{p_{j,x}}{\Delta})^2 + (\frac{p_{i,y}}{\Delta} + 1 - \frac{p_{j,y}}{\Delta})^2}$$

From the triangle inequality property, such that a, b and c are the lengh of the triangle edges

$$c \leq a + b$$



We can simplify the range of d^* to

$$\sqrt{(\frac{p_{i,x}}{\Delta} - \frac{p_{j,x}}{\Delta})^2 + (\frac{p_{i,y}}{\Delta} - \frac{p_{j,y}}{\Delta})^2} - \sqrt{2} \leq d^* \leq \sqrt{(\frac{p_{i,x}}{\Delta} - \frac{p_{j,x}}{\Delta})^2 + (\frac{p_{i,y}}{\Delta} - \frac{p_{j,y}}{\Delta})^2} + \sqrt{2}$$
$$\frac{d}{\Delta} - \sqrt{2} \leq d^* \leq \frac{d}{\Delta} + \sqrt{2}$$

Hence, the error of d^* is $2\sqrt{2}$ at most.
Therefore,

$$2n\sqrt{2}\Delta = \varepsilon OPT$$
$$\Delta = \frac{\varepsilon OPT}{2n\sqrt{2}}$$

(ii)

Let P and P^* denote set of edges from the optimal solution and the PTAS algorithm respectively and we know that $length^*(T) \geq length^*(T^*)$, then we have

$$\sum_{p_i,p_j \in P^*} d^*_{ij} \leq \sum_{p_i,p_j \in P} d^*_{ij}$$

Thus, we can derive

$$\begin{aligned}
length(T^*) &= \sum_{p_i,p_j \in P^*} d_{ij} \\
&\leq \sum_{p_i,p_j \in P^*} \Delta(d^*_{ij} + \sqrt{2}) \\
&\leq \Delta \sum_{p_i,p_j \in P^*} (d^*_{ij} + \sqrt{2}) \\
&\leq \Delta \sum_{p_i,p_j \in P} (d^*_{ij} + \sqrt{2}) \\
&\leq \Delta \sum_{p_i,p_j \in P} (\frac{d_{ij}}{\Delta} + 2\sqrt{2}) \\
&\leq \sum_{p_i,p_j \in P} d_{ij} + \Delta \sum_{p_i,p_j \in P} 2\sqrt{2} \\
&\leq length\,(T) + \Delta 2|P|\sqrt{2} \\
&\leq OPT + \left(\frac{\varepsilon OPT}{2n\sqrt{2}}\right) 2n\sqrt{2} \\
&\leq (1+\epsilon)OPT
\end{aligned}$$

(iii)

Let m^* denote the new boundary of the coordinate after rounding p_x, p_y to p^*_x, p^*_y and we also know that

$$\begin{aligned}
m &= max(p_x, p_y) \\
OPT &\geq 2m
\end{aligned}$$

Thus

$$\frac{m}{\Delta} \leq m^* \leq \frac{m}{\Delta} + 1$$

Then, we can derive the running time

$$\begin{aligned} m* &\leq \frac{m}{\Delta} + 1 \\ &\leq \frac{m2n\sqrt{2}}{\epsilon OPT} + 1 \\ &\leq \frac{m2n\sqrt{2}}{\epsilon 2m} + 1 \\ &\leq \frac{n\sqrt{2}}{\epsilon} + 1 \end{aligned}$$

Therefore, the running time is

$$\begin{aligned} O\left(nm^*\right) &= O\Big(n\frac{n\sqrt{2}}{\epsilon} + 1\Big) \\ &= O\Big(\frac{n^2\sqrt{2}}{\epsilon}\Big) \end{aligned}$$

AIII.3

(i)

Because we know that $ALG(G, \epsilon) \in \mathbb{N}$, so that if we can find such ϵ that the algorithm yields

$$OPT - 1 < ALG\left(G, \epsilon\right) \leq OPT$$

Then, we can get OPT in polynomial time.

In order to get such ϵ, we will derive

$$\begin{aligned} ALG\left(G, \epsilon\right) &> OPT - 1 \\ &> \left(1 - \frac{1}{OPT}\right) OPT \end{aligned}$$

Hence we can get OPT if we choose $\epsilon < \frac{1}{OPT}$ and we also know that the algorithm uses $ALG(G, \epsilon)$ as a subroutine.

Therefore, there is no such FPTAS exist.

(ii)

The proof above indeed implies that there is no PTAS such a problem because we know that a PTAS algorithm also computes a $(1-\epsilon)$-approximation for the problem and if we choose $\epsilon > \frac{1}{OPT}$ as the proof above then, the PTAS algorithm will yield OPT in polynomial time of n.

Therefore there is no PTAS exist anymore.