COMP4600/8460 - Advanced Algorithms **S2 2022**

Assignment 2

[Xueer Zhangsong] ([u7079320]) Date: August 28, 2022

Note: *Plagiarism is strictly prohibited. Even though you may discuss with classmates, you should write your homework assignment by yourself.*

1 Questions

Q1) Prove Strong Duality Theorem (5 Marks)

Q2) Implement primal-dual algorithm $\mathcal{A}_{\mathsf{PDsetcover}}$ in Python/Java/C. Evaluate the approximation ratio on random SetCover instances, and compare the approximation ratio with greedy algorithm $\mathcal{A}_{\mathsf{setcover}}$. To generate a random SetCover instance, you may assign a cost to each cover by a uniform probability distribution, and make each item coverable by a particular cover with a probability = 0.5. Plot and discuss the results of approximation ratios here in this report, and upload your code separately on Wattle (10 Marks)

Q3) Show 3SAT $\preceq$ SubsetSum $\preceq$ Knapsack (10 Marks)

Q4) Study BinPacking problem:

Q4.1) Show that there is a 2-approximation for BinPacking (5 Marks)

Q4.2) Show that there is no $(\frac{3}{2} - \epsilon)$-approximation for BinPacking (5 Marks)

2 Answers

2.1 Q1

Q1.1) Prime and dual problem

Prime: Maximize $C^T x$, s.t. $Ax \leq b$ and $x \geq 0$, $A \in \mathbb{R}^{n*m}$, $x \in \mathbb{R}^n$

Dual: Minimize $b^T y$, s.t. $Ay \geq C$ and $y \geq 0$, $A \in \mathbb{R}^{m*n}$, $y \in \mathbb{R}^m$

Q1.2) Prove Farkas' lemma

A convex cone generated by n vectors $a_1 \dots a_n \in \mathbb{R}^m$ can be form as $\{x_1a_1 + \dots + x_na_n | x_1 \dots x_n \geq 0\}$.

Which can be simplify to $\{Ax | x \in \mathbb{R}^n, x \geq 0\}$.

There exist a vector b, $b \in \mathbb{R}^n$:

(1) When b is in the cone:

There must be a $x \in \mathbb{R}^n$, that achieve Ax=b, when b is the maximum solution that A can generated, we can have:

$$Ax \leq b$$

(2) When b is not in the cone:

Which means that the cone and b on is on the two sides of a random hyperplane which is define by a set of $x \in \mathbb{R}^m$. Where $y \in \mathbb{R}^m$ is some normal vector of this plane, the inter-product of $y^t x$ is equal to 0. The cone is at the positive side of the plane, and b is at the negative side. Thus, we can have:

$$yA \geq 0$$
$$b^T y < 0$$

Q1.2) Prove Strong duality

Given prime optimal solution x*. Than CT x* is the optimal value of the prime problem.

When $\epsilon > 0$, $\nexists x \geq 0$, such that:

$$Ax \leq b \tag{1}$$
$$C^T x \geq C^T x^* + \epsilon \tag{2}$$

From (1) - (2), we can get:

$$\begin{pmatrix} A \\ -C^T \end{pmatrix} x \leq \begin{pmatrix} b \\ -C^T x^* - \epsilon \end{pmatrix} \tag{3}$$

Let $\hat{A} = \begin{pmatrix} A \\ -C^T \end{pmatrix}$, and $\hat{b} = \begin{pmatrix} b \\ -C^T x^* - \epsilon \end{pmatrix}$.

According to (3), we can get:

$$\hat{A}x \leq \hat{b} \tag{4}$$

Because x has no non-negative solution, it doesn't hold Farkas' lemma (1). This implies that Farkas' lemma (2) is hold.

Thus, there exist $\hat{y} = \begin{pmatrix} y \\ z \end{pmatrix} \in \mathbb{R}^{m+1}, z \in \mathbb{R}$ where:

$$\begin{aligned} \hat{y}^T \hat{A} &\geq 0 \\ \begin{pmatrix} y^T & z \end{pmatrix} \begin{pmatrix} A \\ -C^T \end{pmatrix} &\geq 0 \\ y^T A - zC^T &\geq 0 \\ y^T A &\geq zC^T \end{aligned} \qquad \begin{aligned} \hat{b}^T \hat{y} &< 0 \\ \begin{pmatrix} b^T & -C^T x^* - \epsilon \end{pmatrix} \begin{pmatrix} y \\ z \end{pmatrix} &< 0 \\ b^T y - z(C^T x^* + \epsilon) &< 0 \\ b^T y &< z(C^T x^* + \epsilon) \end{aligned} \tag{5}$$

z is a real number:

$$\begin{aligned} y^T A &\geq zC \\ A^T y &\geq zC \end{aligned}$$

When $\epsilon = 0$, $\exists x \geq 0$, such that:

$$\begin{aligned} Ax &\leq b \\ C^T x &\geq C^T x^* \\ \text{from (6) and (7):} & \\ \begin{pmatrix} A \\ -C^T \end{pmatrix} x &\leq \begin{pmatrix} b \\ -C^T x^* \end{pmatrix} \end{aligned} \tag{6}$$

Thus, Farkas' lemma (1) holds and (2) doesn't holds. Which indicates that:

$$\hat{y}^T \begin{pmatrix} A \\ -C^T \end{pmatrix} < 0 \tag{7}$$

$$\begin{pmatrix} b \\ -C^T x^* \end{pmatrix} \hat{y} \geq 0 \tag{8}$$

By (8), we are able to get $b^T y > z C^T x^*$ From (5) and (8):

$$\begin{aligned} b^T y &< z(C^T x^* + \epsilon) \\ b^T y &\geq z C^T x^* \\ \text{Rearrange:} & \\ z C^T x^* \leq b^T y &< z(C^T x^* + \epsilon) \\ z C^T x^* &< z(C^T x^* + \epsilon) \\ 0 &< z\epsilon \end{aligned} \tag{9}$$

In the first case, $\epsilon > 0$, in order to meet (9), the value of z has to be greater than 0.
Since $z > 0$, from (5), we can get:

$$A^T \frac{y}{z} \geq C \qquad\qquad b^T \frac{y}{z} < C^T x^* + \epsilon \tag{10}$$

Thus, $\frac{y}{z}$ is a feasible solution of the dual. When $\epsilon \to 0$, $\frac{y}{z}$ is the optimal solution of the dual problem. Then, we can get:

$$b^T y^* < C^T x^* + \epsilon \tag{11}$$

According to the dual problem:

$$x \geq 0 \tag{12}$$

$$y \geq 0 \tag{13}$$

$$Ay \geq C \tag{14}$$

$$Ax \leq b \tag{15}$$

From (12) and (14): (16)

$$Ayx \geq Cx \tag{17}$$

$$Cx \leq y^T Ax \tag{18}$$

From (13) and (15): (19)

$$y^T Ax \leq y^T b \tag{20}$$

From (18) and (20): (21)

$$Cx \leq y^T Ax \leq y^T b \tag{22}$$

$$Cx \leq y^T b \tag{23}$$

From (11) and (23), we can get:

$$Cx^* \leq b^T y^* < C^T x^* + \epsilon \tag{24}$$

This inequality holds when, $Cx^* = b^T y^*$

Thus the strong duality of linear programming is proved.

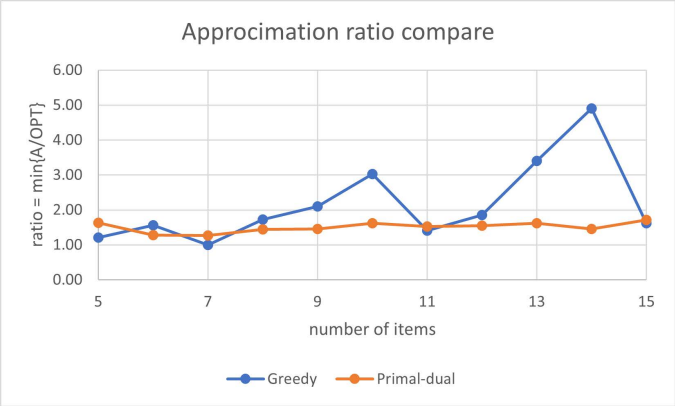
2.2 Q2

Firstly, since set cover is a minimum problem, the approximation ratio should be define as:

$\alpha = max \frac{A}{OPT}$

For each item number, I calculate the approximation ratio for both algorithms 20 times using different random instance and output the maximum one according to the definition.

The following line chart compares the maximum approximation ratio of the greedy algorithm and the primal-dual algorithm when the number of items is between 5 and 15.



From the above chart, we can find that when the number of items gradually increases, the performance of the greedy algorithm is extremely unstable depending on the random set cover instance. If the items in the random set cover instance conform to a certain order, even if the number of items is large, the greedy algorithm can also give results close to OPT. But many times, when the greedy algorithm calculates a random set cover instance, the result is often very different from the OPT.

In contrast, the approximation ratio of the primal-dual algorithm is relatively stable when the number of items gradually increases, and shows an increasing trend.

Therefore, even if the greedy algorithm performs well in some specific cases, the gap between its best and worst cases is too large, and its results are extremely unstable as the number of items increases. Therefore, for the set cover problem, the more stable primary-dual algorithm is a better choice.

2.3 Q3

Q1.1) Reduce subset sum to Knapsack

We define the subset sum problem as: there are n non-negative numbers s_i in set S_1 and a target value T. And we aim to find $\sum_{i \in S} s_i = T$.

For Knapsack problem: for n items, there are non-negative weights w_i in set S_2, a maximum weight W, and non-negative values v_i in set S_3. And we aim to meet the constrain $\sum_{i \in S_2} w_i \leq W$ and also maximize the sum of the value of those item.

According to the basic definition of the Knapsack problem. We can set a value V, which represent the bottom line of the sum of the values. Thus, the purpose of the Knapsack problem can be changed to find a subset A of n items that:

$$\sum_{i \in S_2} w_i \leq W$$
$$\sum_{i \in S_3} v_i \geq V$$

Next, assume there is a Knapsack problem and a subset sum problem achieve the following condition:

$$\begin{cases} w_i = v_i = s_i \\ W = V = T \end{cases}$$

According to the above conditions, we can derive the following inequalities:

$$\sum_{i \in S_2} w_i \leq W \Longleftrightarrow \sum_{i \in S_1} s_i \leq T \tag{25}$$
$$\sum_{i \in S_3} v_i \geq V \Longleftrightarrow \sum_{i \in S_1} s_i \geq T \tag{26}$$

From (25) and (26), we can get:

$$\sum_{i \in S} s_i = T \tag{27}$$

Equality (27) is the same as the expression of the subset sum problem.

For this case, if the subset sum problem has a solution, than the corresponding Knapsack problem can be solved, and vise versa. This also means that if and only if the answer for subset sum is "yes", then the answer for Knapsack is "yes", otherwise it will be "no".

And this reduction can be done in polynomial time.

Q1.2) Reduce 3SAT to subset sum

Assume that we define a 3SAT problem with formula of n variables, from $x_1 \ldots x_n$, and m clauses, from $c_1 \ldots c_m$. And a subset sum problem with formula of non-negative numbers w_i in set S and a target value W.

We need to show that there is a polynomial time transformation from 3SAT to subset sum.

Firstly, we rearrange the 3SAT formula.

For each variable x_i, we define two numbers t_i and f_i, which represent x_i is True and False respectively.

Afterwards, we construct t_i and f_i as:

$$t_i = 10^{m+i-1} + \sum_{j=c_j contains x_i} 10^{j-1}$$
$$f_i = 10^{m+i-1} + \sum_{j=c_j contains \bar{x_i}} 10^{j-1}$$

If n is 4 and m is 3, the Boolean equation is:

$$\phi = (x_1 \vee x_2 \vee x_3) \wedge (\bar{x_2} \vee x_3 \vee x_4) \wedge (x_1 \vee \bar{x_2} \vee \bar{x_3})$$

the variables will end up as a table:

	10^6	10^5	10^4	10^3	10^2	10^1	10^0
variables	x_1	x_2	x_3	x_4	c_1	c_2	c_3
t_1	1	0	0	0	1	0	1
f_1	1	0	0	0	0	0	0
t_2	0	1	0	0	1	0	0
f_2	0	1	0	0	0	1	1
t_3	0	0	1	0	1	1	0
f_3	0	0	1	0	0	0	1
t_4	0	0	0	1	0	1	0
f_4	0	0	0	1	0	0	0

The 1s and 0s in the 0-2 digit indicate that whether the variable x_i or $\bar{x_i}$ is in clauses 0-2 or not. The 1s and 0s in the first 4 digit is aim to show which variable t_i and f_i is representing.

Assume we have a satisfying assignment, it can be represented as follow:

	10^6	10^5	10^4	10^3	10^2	10^1	10^0
variables	x_1	x_2	x_3	x_4	c_1	c_2	c_3
t_1	1	0	0	0	1	0	1
f_1	1	0	0	0	0	0	0
t_2	0	1	0	0	1	0	0
f_2	0	1	0	0	0	1	1
t_3	0	0	1	0	1	1	0
f_3	0	0	1	0	0	0	1
t_4	0	0	0	1	0	1	0
f_4	0	0	0	1	0	0	0
SA	1	1	1	1	> 0	> 0	> 0

Where in the number SA (stands for satisfying assignment), the digit of 10^{j-1} indicate the number of true value in clause j, the maximum is 3 and based on 3SAT formula this is a non-zero number. The digit of 10^{m+i-1} is 1, which means for each variable x_i, either t_i or f_i is included (base on the formula of 3SAT every variable no matter true or false in the variable set must appear at least once in the equation).

Next we set target value W' of subset sum equal to SA. Since W' should be a fixed number, we need to settle the number at 10^{j-1} digit.

We can achieve that by adding two numbers a_j and b_j for each clause, where x and y are positive constant:

$$a_j = 10^{j-1}$$
$$b_j = 10^{j-1}$$

Let W' = $\sum_i^n 10^{m+i-1} + K \sum_j^m 10^{j-1}$.

This number K indicate the selected 1s at the digit of 10^{j-1}. let s_1 indicate the number of 1s selected in the t_i and f_i section, and s_2 be the 1s selected in the additional numbers section.

Let $K = 3$.

Since there are only 2 additional numbers for each clause, so $max\{s_2\} = 2$. In order to achieve 3 selects, s_1 has to achieve $s_1 \geq 1$. In this way, we have fixed the value of the last three digits while ensuring the satisfying assignment of 3SAT.

In addition, in theory, for each clause, we can add more than 2 numbers $(a_j, b_j, c_j \ldots)$, and its constant can also be greater than or equal to 1 $(a_j = 2 \times 10^{j-1})$, as long as for all $s_1 = 1, 2, 3$ there exist a combination of s_1 and s_2 meet that $K = s_1 + s_2$ and $K \leq 9$ (we should not have carries).

When $K = 3$, and add a_j and b_j for each clause, we can have a table:

	10^6	10^5	10^4	10^3	10^2	10^1	10^0
variables	x_1	x_2	x_3	x_4	c_1	c_2	c_3
t_1	1	0	0	0	1	0	1
f_1	1	0	0	0	0	0	0
t_2	0	1	0	0	1	0	0
f_2	0	1	0	0	0	1	1
t_3	0	0	1	0	1	1	0
f_3	0	0	1	0	0	0	1
t_4	0	0	0	1	0	1	0
f_4	0	0	0	1	0	0	0
a_1	0	0	0	0	1	0	0
b_1	0	0	0	0	1	0	0
a_2	0	0	0	0	0	1	0
b_2	0	0	0	0	0	1	0
a_3	0	0	0	0	0	0	1
b_3	0	0	0	0	0	0	1
W'	1	1	1	1	3	3	3

Treat all the numbers in the table as decimal numbers.

For subset sum problem. Let set S = $\{t_1, f_1 \ldots t_4, f_4, a_1, b_1 \ldots a_3, b_3\}$ and W = W'.

Since W is a satisfying assignment of 3SAT, as a target of subset sum, there must exist a subset in set S that it's items can sum up and equal to W. This also means that if and only if the answer for 3SAT is "yes", then the answer for subset sum is "yes".

And All of the above reduction algorithm can be done using for loop, thus is can be done in polynomial time.

2.4 Q4

Q4.0) Defining the Bin Packing problem

Firstly, we define the Bin Packing problem as follow:
Given n pieces, with size S = $[s_1, s_2 \ldots s_n]$, such that $\forall a_i \in (0, 1)$.
Given a bunch of bins, the maximum size a bin can hold is 1.
We aim to find the minimum number of bins that can hold all of the given pieces.

Q4.1) 2-approximation for BinPacking

Define the Next-fit algorithm to solve the bin packing problem as follow:

$bins \leftarrow 0$

$remainSpace \leftarrow 1$

For every pieces i:

if $remainSpace \geq S[i]$:

$remainSpace = remainSpace - S[i]$

else:

$bins = bins + 1$

$remainSpace = 1 - S[i]$

Given a situation where there are bins are used in a bin packing problem and the sum of size in each bins are $b_1 \dots b_k$. Assume that the size of every piece in S is $0.5 + \epsilon$, where $\epsilon \to 0$.

Consider any two adjacent bins m-1 and m, where $1 \leq m \leq k$. According to the Next-fit algorithm above, item will be allocated to the second bin if and only if the remaining space of the first bin is not enough. Thus the items sums of the adjacent two bins must greater than 1:

$$b_{m-1} + b_m > 1$$

From assumption, the remaining space of two bins is:

$$\begin{aligned} remains &= 2 - (b_{m-1} + b_m) \\ &= 2 - (0.5 + \epsilon + 0.5 + \epsilon) \\ &= 1 - 2\epsilon \end{aligned}$$

When $\epsilon \to 0$, the $remains \to 1$. The remains can never be greater than one according to the Next-fit algorithm.

Thus, according to the above derivation, in general bin packing cases, using the Next-fit algorithm, we can conclude that the remains of the two adjacent bins is at most 1. Moreover, this happens between every adjacent bins along the bin list. Which mean at most half of the bins space are empty.

If applying the Next-fix algorithm will use M bins, and at most half of the bins space are empty. We can have:

$$\frac{M}{2} < \sum_i^n s_i$$

Because the sum of the items size is the lower bound of the optimal solution (according to the definition of the bin packing problem, $0 < s_i < 1$, $\max b_m = 1$), we can have:

$$\begin{aligned} \sum_i^n s_i &\leq OPT \\ \frac{M}{2} &\leq OPT \\ M &\leq 2OPT \end{aligned}$$

Thus, there is a 2-approximation for BinPacking

Q4.2) no $(\frac{3}{2} - \epsilon)$-approximation for bin packing

Q4.2.1) Prove partition problem is in NP

The definition of the partition problem as follow:

Given a set of numbers, and we aim to find if this set can be partition into two subset, where the sum of numbers in each subset are the same.

Given a certificate of the partition problem $\{S, A_1, A_2\}$. Where S is the given set of the partition problem. A_1 and A_2 are the two subset of S.

The verify algorithm V is as follow:

$n \leftarrow 0$

$s_1 \leftarrow 0$

$s_2 \leftarrow 0$

for items i in $A_1 \cup A_2$:

    if $i \in S$:

        $n = n + 1$

if $n = |S|$:

    for items x in A_1:

        $s_1 = s_1 + x$

    for items y in A_2:

        $s_2 = s_2 + y$

    if $s_1 = s_2$:

        return True

return False

The algorithm V run in $\log(n)$ time.

Thus, the certificate of partition problem can be verify in polynomial time. Since the length of the certificate also has a polynomial length.

The partition problem is in NP.

Q4.2.2) Prove no $(\frac{3}{2} - \epsilon)$-approximation for BinPacking

Assuming there is an approximation algorithm A can have a $(\frac{3}{2} - \epsilon)$-approximation for bin packing. Since $(\frac{3}{2} - \epsilon) \to 1$, this algorithm actually is about to give the optimal solution for bin packing problem. And this algorithm A runs in a polynomial time.

Consider an algorithm B, which partition the n pieces in bin packing problem into two sets. Where the sum of all the piece in each set is equal to $\frac{1}{2}\sum_i^n s_i$.

The algorithm will return "yes" if these n piece and be packed into 2 bins, where each bin contains the sizes of $\frac{1}{2}\sum_i^n s_i$. And returns "no" otherwise.

The above algorithm B can run in a polynomial time.

This indicate that, bin packing problem can reduce to partition problem using a polynomial algorithm B.

And since A can find a$(\frac{3}{2} - \epsilon)$-approximation solution for bin packing, then with the reduction algorithm B, using both B and A can find a optimal solution for partition problem in polynomial time. This shows that the partition problem is in p.

However, as we prove above, partition problem is in NP.

Since $NP \neq P$.

There is a contradiction. Thus such approximate algorithm A doesn't exist.

There is no $(\frac{3}{2} - \epsilon)$-approximation for bin packing

...

References

[Vazirani] V. Vazirani, Approximation Algorithms, Springer, (2003).