

COMP2007 Assignment 5 Report

Jiashu Wu 460108049 jiwu0083

Question 1

Question 1 (a)

Question 1 (a) i. Describe your reduction

For this reduction, I set the four parameters of Knapsack problem as following:

- Value: payoffs array
- Weight: payoffs array
- Capacity $W = \frac{1}{2} * \sum_{i=1}^{n_2} payoff f_i$
- Target $V = c * \frac{1}{2} * \sum_{i=1}^{n_2} payoff f_i$

Question 1 (a) ii. The running time of the algorithm

Since the length of the payoffs array is n_2 , thus calculating the sum costs $O(n_2)$ work, and assigning the payoffs array to variable 'Value' and 'Weight' also takes $O(n_2)$ work. Hence, assigning all these four parameters takes $4 * O(n_2) = O(n_2)$ time in total. Hence, the running time of the reduction is $O(n_2)$.

Therefore, the running time of the whole algorithm is

$$\begin{aligned} O(n_2) + O(f_1(n_2, W, V)) &= O(n_2) + O(f_1\left(n_2, \frac{1}{2} * \sum_{i=1}^{n_2} payoff f_i, c * \frac{1}{2} * \sum_{i=1}^{n_2} payoff f_i\right)) \\ &= O(n_2 + f_1(n_2, \frac{1}{2} * \sum_{i=1}^{n_2} payoff f_i, c * \frac{1}{2} * \sum_{i=1}^{n_2} payoff f_i)), \end{aligned}$$

which is the running time of the reduction plus the running time of the Knapsack problem with new parameters.

Question 1 (a) iii. Prove the correctness of the algorithm

Prove that a YES instance of Knapsack problem will be a YES instance for C-Fair Sharing problem

Since we have already set the capacity as $\frac{1}{2} * \sum_{t_i \in T} p_i$, the target as $c * \frac{1}{2} * \sum_{t_i \in T} p_i$, both the weight and value as payoff values array, and since for a YES instance of Knapsack problem, it satisfies that the sum of weight of selected items is less than or equal to the capacity, and it also satisfies that the sum of value of selected items is greater than or equal to the target, therefore, this YES instance of Knapsack problem satisfies exactly the two constraints of the C-Fair Sharing problem, which is

$$c * \frac{1}{2} * \sum_{t_i \in T} p_i \leq \sum_{t_i \in T_1} p_i \leq \frac{1}{2} * \sum_{t_i \in T} p_i$$

Therefore, this YES instance of Knapsack problem will also be a YES instance of the C-Fair Sharing problem.

Prove that a NO instance of Knapsack problem will be a NO instance of C-Fair Sharing problem

Since for a NO instance of Knapsack problem, it doesn't satisfy at least one of the following two constraints: the sum of weight of a subset of items is less than or equal to the capacity, or the sum of the value of a subset of items is greater than or equal to the target. Therefore, if we use payoff values array as both weight and value, use $\frac{1}{2} * \sum_{t_i \in T} p_i$ as the capacity, and use $c * \frac{1}{2} * \sum_{t_i \in T} p_i$ as the target, a NO instance of Knapsack tells us that at least one of the C-Fair Sharing constraints will not be satisfied. Therefore, a No instance of Knapsack problem will also be a NO instance of the C-Fair Sharing problem.

Therefore, the reduction is correct.

Question 1 (b)

Question 1 (b) i. Describe your reduction

For this reduction, I set the two parameters of C-Fair Sharing problem as following:

- $C = 1$
- Payoffs = a difference array which contains $\text{task_pair}[0] - \text{task_pair}[1]$ for all n_3 task pairs.

Question 1 (b) ii. The running time of the algorithm

Since there are n_3 task pairs, thus calculating the difference array takes $O(n_3)$ time, and assigning c to one takes $O(1)$. Therefore, the running time of the reduction is $O(n_3) + O(1) = O(n_3)$.

Therefore, the running time of the whole algorithm is

$$\begin{aligned}
 & O(n_3) + O(n_2) + O(f_1(n_3, W, V)) \\
 &= O(n_3) + O(f_1(n_3, \frac{1}{2} * \sum_{(a_i, b_i) \in F} (a_i - b_i), \frac{1}{2} * \sum_{(a_i, b_i) \in F} (a_i - b_i))) \\
 &= O(n_3 + f_1(n_3, \frac{1}{2} * \sum_{(a_i, b_i) \in F} (a_i - b_i), \frac{1}{2} * \sum_{(a_i, b_i) \in F} (a_i - b_i)))
 \end{aligned}$$

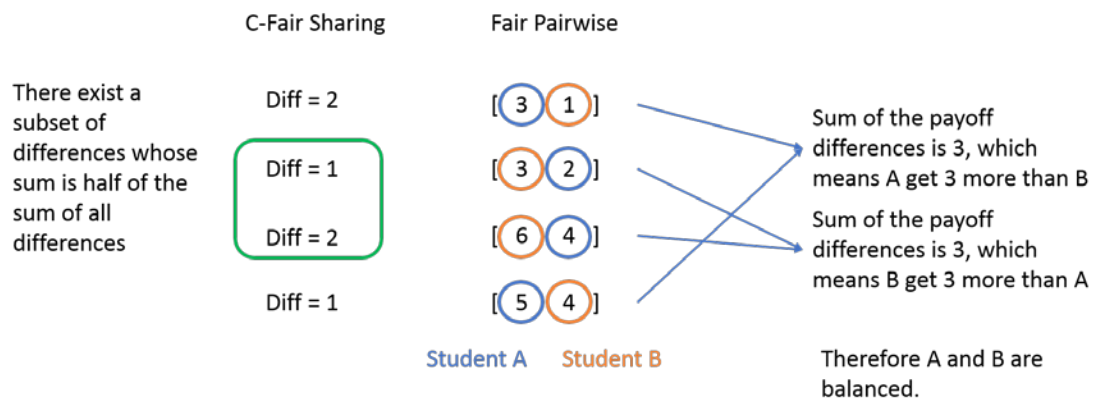
which is the running time of the reduction plus the running time of the C-Fair Sharing problem with new parameters.

Question 1 (b) iii. Prove the correctness of the algorithm

Prove that a YES instance of C-Fair Sharing problem will be a YES instance of Fair Pairwise problem

Since we have allocated c as 1 and we have used the differences array as the payoff for C-Fair Sharing problem, thus a YES instance of C-Fair Sharing problem satisfies that there exists a subset of differences, the sum of them is half of the sum of all differences, mathematically, $\sum_{diff \text{ in subset}} diff = \frac{1}{2} \sum_{all \text{ diff}} diff$.

And if there exists a Fair Pairwise solution, then there must exist a subset of differences whose sum is half of the sum of all differences, and the reason is that if we allocate the left hand side task of k pairs to student A, the right hand side task to student B, and the sum of the payoff differences between left task and right task is S , then in order to distribute the tasks evenly among two students, we need to have rest of the task pairs whose sum of payoff differences between left task and right task are exactly the same as S , otherwise we can't distribute it evenly. And since this instance is a YES instance of C-Fair Sharing problem with $c = 1$ and differences array as the payoff array, thus there exists a subset of differences whose sum is half of the sum of all differences, thus this instance is also a YES instance of Fair Pairwise problem.



	C-Fair Sharing	Fair Pairwise
There isn't a subset of differences whose sum is half of the sum of all differences	Diff = 2	[3 1]
	Diff = 2	[3 1]
	Diff = 2	[6 4]
	Diff = 1	[5 4]

Prove that a Yes instance of Fair Pairwise problem will be a YES instance of C-Fair Sharing problem

As explained in the previous part, if there is a YES instance of Fair Pairwise problem, then there must exist a subset of differences whose sum is half of the sum of all difference values. And since we set $c = 1$ and use the payoff array as the differences array, thus this Yes instance of Fair Pairwise satisfies the constraint $\sum_{task \text{ in } T} payoff = \frac{1}{2} \sum_{all \text{ task in } T} payoff$. Therefore, the YES instance of Fair Pairwise problem is also a YES instance of C-Fair Sharing problem since the constraints of the C-Fair Sharing problem is satisfied.

Therefore, the reduction is correct.

Question 1 (c)

Question 1 (c) i. Describe your reduction

For this reduction, I set the parameter of Fair Pairwise problem as following:

- An array which contains the following three kinds of entries:
If it goes left, then $[0, -\text{distance}]$
If it goes right, then $[\text{distance}, 0]$
After adding all entries above, add this extra entry at the end of the array:
 $[2 * c * \sum(\text{distance without sign}) - \sum(\text{distance with sign}), 0]$, mathematically,

$$[2 * c * \sum_{i=1}^{n_4} |\text{dist}_i| - \sum_{i=1}^{n_4} \text{dist}_i, 0]$$

And then use this array as the parameter of Fair Pairwise problem.

Question 1 (c) ii. The running time of the algorithm

Since there are n_4 commands, thus building this new array takes $O(n_4 + 1) = O(n_4)$ time. Therefore, the running time of the reduction is $O(n_4)$.

Therefore, the running time of the whole algorithm is

$$\begin{aligned} & O(n_4) + O(n_3) + O(n_2) + O(f_1(n_4, W, V)) \\ &= O(n_4) + O(f_1(n_4, W, V)) = O(n_4 + f_1(n_4, W, V)) \\ \text{where } W = V &= \frac{1}{2} \sum_{i=1}^{n_4} |\text{dist}_i| + 2 * c * \sum_{i=1}^{n_4} |\text{dist}_i| - \sum_{i=1}^{n_4} \text{dist}_i \end{aligned}$$

which is the running time of the reduction plus the running time of the Fair Pairwise problem with new parameters.

Question 1 (c) iii. Prove the correctness of the algorithm

Prove that a YES instance of Fair Pairwise problem will be a YES instance of C-Fixed Distance problem

Since we use the distance array as the parameter of Fair Pairwise problem, and there is a YES instance of Fair Pairwise problem, therefore, it means that the distance pairs of this instance can be evenly distributed to two parts and for each task pair one of the task belongs to chosen subset, and another task belongs to not-chosen subset.

Since $c \geq 0.5$, therefore the non-zero term in last entry of the distance array must not belong to that chosen subset of distance entries, otherwise the equation

$\sum_{\text{dist in subset}} \text{dist with sign} = c * \sum_{\text{all dist}} \text{dist without sign}$ will never hold. Since that is a YES instance of Fair Pairwise problem, we get











$$-d_{\text{leftchoose}} + d_{\text{rightchoose}}$$

$$= -d_{leftnotchoose} + d_{rightnotchoose} + 2 * c * (d_{leftchoose} + d_{rightchoose} + d_{leftnotchoose} + d_{rightnotchoose}) - (d_{rightnotchoose} + d_{rightchoose} - d_{leftnotchoose} - d_{leftchoose}),$$

which is

$d_{leftchoose} + d_{rightchoose} = c * (d_{leftchoose} + d_{rightchoose} + d_{leftnotchoose} + d_{rightnotchoose}) = c * \sum_{i=1}^{n_4} dist_i$ after transposition the terms, which is exactly the constraint of C-Fixed Distance problem. Hence, a YES instance of Fair Pairwise problem is also a YES instance of C-Fixed Distance problem.

Note: $d_{leftchoose}$ in the above formula means $\sum_{all\ the\ left\ distance\ we\ choose\ to\ go\ diff_i}$, the sum of the distance of all left commands we choose to go.

Fair Pairwise		C-Fixed Distance	
	LEFT 1		Sum(distance without sign) = 1 + 2 + 3 + 6 = 12
	LEFT 2		Sum(distance with sign) = -1 + -2 + 3 + 6 = 6
	RIGHT 3		C = 7/12
	RIGHT 6		Extra Entry = 2*c*sum(distance without sign) - sum(distance with sign) = 8
	Extra Entry		
Student B Student A		Not in travel subset In travel subset	
	The extra entry must not belong to the subset since $c \geq 0.5$		

Prove that a YES instance of C-Fixed Distance problem will be a YES instance of Fair Pairwise problem

Since we used the distance array as the task pair array and there is a YES instance of C-Fixed Distance problem, therefore this YES instance must satisfy $d_{leftchoose} + d_{rightchoose} = c * (d_{leftchoose} + d_{rightchoose} + d_{leftnotchoose} + d_{rightnotchoose}) = c * \sum_{i=1}^{n_4} dist_i$, which means we have a way to distribute the task pairs into two parts evenly, and for each task pair one of the tasks belongs to group A and another task belongs to group B, and this is exactly the constraints of Fair Pairwise problem. Hence, a YES instance of C-Fixed Distance problem is also a YES instance of Fair Pairwise problem.

Therefore, the reduction is correct.

Question 2

Question 2 (a). Prove that the Fair Pairwise decision problem is NP-complete

Before proving we assume that:

- the polynomial time reduction from C-Fixed Distance problem to Fair Pairwise problem is correct.
- C-Fixed Distance decision problem is in NP-complete.

To prove that Fair Pairwise problem is NP-complete, I will prove the following two things:

- Fair Pairwise problem is in NP
- Fair Pairwise problem is in NP-hard.

Prove Fair Pairwise problem is in NP

Certificate: An array containing several (a_i, b_i) pairs, and an array that records which a_i is assigned to student A.

Certifier:

1. Calculate the sum of all a_i which are assigned to student A and all b_i whose corresponding a_i in the pair is not assigned to student A.
2. Calculate the sum of all b_i whose corresponding a_i in the pair is assigned to student A and all a_i which is not assigned to student A.
3. Compare these two sums, if they are equal, return TRUE, otherwise return FALSE.

Running time of the certifier

Since there are n_3 pairs in total, and the length of the array is at most n_3 , thus step 1 and step 2 takes at most $O(n_3 * n_3) = O(n_3^2)$ time. Therefore, the certifier runs in $O(n_3^2)$ time.

Therefore, there exists a polynomial time certifier.

Therefore, Fair Pairwise problem is in NP.

Prove Fair Pairwise problem is in NP-hard

Since there exist a NP-complete problem (C-Fixed Distance problem), which can reduce to Fair Pairwise problem in polynomial time, therefore, Fair Pairwise is in NP-hard.

Since Fair Pairwise problem is in both NP and NP-hard, hence it is in NP-complete.

Question 2 (b). Prove that the C-Fair Sharing decision problem is NP-complete

Before proving we assume that:

- the polynomial time reduction from Fair Pairwise problem to C-Fair Sharing problem is correct.
- Fair Pairwise decision problem is in NP-complete. (Assume proved correctly by Q2(a))

To prove that C-Fair Sharing problem is NP-complete, I will prove the following two things:

- C-Fair Sharing problem is in NP
- C-Fair Sharing problem is in NP-hard.

Prove C-Fair Sharing problem is in NP

Certificate: A given value of c , a payoffs array and a subset of tasks.

Certifier:

1. Take the sum of all payoff values, we denote it as S_1 .
2. Take the sum of all payoff values of the tasks belongs to the given subset, we denote it as S_2 .
3. If $c * 0.5 * S_1 > S_2$, return FALSE
4. If $0.5 * S_1 < S_2$, return FALSE
5. Otherwise return TRUE.

Running time of the certifier

Since the length of the payoffs array is n_2 , and the length of the subset array is at most n_2 , therefore step 1 takes $O(n_2)$ time, and step 2 takes $O(n_2^2)$ time. Step 3 to 5 all takes $O(1)$ time, therefore, the running time of the certifier is $O(n_2) + O(n_2^2) = O(n_2^2)$ time.

Therefore, there exists a polynomial time certifier.

Therefore, C-Fair Sharing problem is in NP.

Prove C-Fair Sharing problem is in NP-hard

Since there exist a NP-complete problem (Fair Pairwise problem), which can reduce to C-Fair Sharing problem in polynomial time, therefore, C-Fair Sharing problem is in NP-hard.

Since C-Fair Sharing problem is in both NP and NP-hard, hence it is in NP-complete.

Question 2 (c). Prove that the Knapsack decision problem is NP-complete

Before proving we assume that:

- the polynomial time reduction from C-Fair Sharing problem to Knapsack problem is correct.
- C-Fair Sharing decision problem is in NP-complete. (Assume proved correctly by Q2(b))

To prove that Knapsack problem is NP-complete, I will prove the following two things:

- Knapsack problem is in NP
- Knapsack problem is in NP-hard.

Prove Knapsack problem is in NP

Certificate: A subset of the original (weight, value) pairs, and the capacity and target value.

Certifier:

1. Take the sum of all weights, if the sum is greater than the capacity, return FALSE.
2. Take the sum of all values, if the sum is less than target, return FALSE.
3. Otherwise, return TRUE.

Running Time Analysis of the certifier

Since there are at most n_1 (weight, value) pairs in total, therefore, the step 1 and the step 2 both takes $O(n_1)$ time. Therefore, the certifier runs in $O(n_1)$ time.

Therefore, there exists a polynomial time certifier.

Therefore, Knapsack problem is in NP.

Prove Knapsack problem is in NP-hard

Since there exist a NP-complete problem (C-Fair Sharing problem), which can reduce to Knapsack problem in polynomial time, therefore, Knapsack problem is in NP-hard.

Since Knapsack problem is in both NP and NP-hard, hence it is in NP-complete.