Package-Delivery

drop table group;
Junjie Wang 517021910093 dreamboy.gns@sjtu.edu.cn
Yikai Yan 517021910404 miku.miku.miku@sjtu.edu.cn
Wentao Qin 517021910483 423445630@qq.com

June 15, 2019

1 Abstract

We construct a graph to represent the city network. Problem 1 is NP. We solved it by modified $dijskra$. Problem 2 and 3 are also NP. We solved them by Sequential Algorithm. Problem 4$\leq_p$ Problem 1, but we solved it by approximation algorithm.

Contents

2 Symbol Table

We list the symbols we use to model the city network, commodities and orders in table 1 and 2. Detailed information is given later.

vertex	$city$	$index$, $attribute(small/substation/hub)$
edge	$dist$	
	$tools$	$depart\_city$, $arrival\_city$, $time\_on\_way$, $depart\_time$, $unit\_amount\_cost$

Table 1: City Network Model

commodity	$index$, $category$, $unit\_weight$
order	$seller\_city$, $purchaser\_city$, $order\_time$, $commodity\_index$, $commodity\_amount$, $emergency$

Table 2: commodity and order Model

3 Problem 1

3.1 Problem Analysis

In problem 1, since cities and transportation tools are not capacitied, orders are independent with each other. Therefore, we need to find an algorithm. The input is a particular order and the and the output is an optimized or approximated scheme for this order.

3.2 Modeling

- Modeling the city network.

  The city network is formulated into a graph. The vertices are the cities and the edges represent the transportation tools between 2 cities and the distance.

vertex u	edge (u, v)
$city$	$tools$
	$dist$

  $tools$ is the set of transportation tools from vertex u to v. $dist$ is the distance from vertex u to v (unit:km). In problem 1, a city is simply an index within set $[656]$ in our model.

- Modeling a transportation tool.

  Any transportation tool $tool \in tools$ is formulated into the set:

$$\{depart\_city\,,\, arrival\_city\,,\, time\_on\_way\,,\, depart\_time\,,\, unit\_amount\_cost\}$$

  $depart\_city$: the index of the city where the tool departs.
  $arrival\_city$: the city where it arrives.
  $time\_on\_way$: the time consumed on the way ($time\_on\_way = Average\_delay\_per\_trip + \frac{dist(u,v)}{speed}$,unit:$min$).
  $depart\_time$: the time it departs at.
  $unit\_amount\_cost$: the cost to transport a unit commodity from u to v ($unit\_amount\_cost = unit\_cost \times dist(u,v)$, unit:$\$/kg$).

- Modeling the commodity.

  A particular kind of commodity is formulated into the set:

$$\{index\,,\, category\,,\, unit\_weight\}$$

  $index$: commodity's index.
  $category$: the category which this kind of commodity belongs to.
  $unit\_weight$: the weight of one unit of such commodity (unit:kg).
  Notice that the information about commodity's unit price is omitted because this has nothing to do with our model.

- Modeling the order.

  Each order is formulated into the set:

$$\{seller\_city\,,\, purchaser\_city\,,\, order\_time\,,\, commodity\_index\,,\, commodity\_amount\,,\, emergency\}$$

  $seller\_city$ and $purchaser\_city$: the index of the 2 cities.
  $order\_time$: the time when the order happens.
  $commodity\_index$: the index of the ordered commodity.

$commodity\_amount$: the number of commodities ordered.

$emergency$: set 1 if it's an emergency order and 0 else.

3.3 Problem Formulation

- Objective Function.

  The function is $f(p)$. p is a path from $seller\_city$ to $purchaser\_city$. We call this function overall evaluation.

  $$f(p) = a \times cost + b \times time.$$

  $cost$ is the total cost of the delivery scheme. And $time$ is the time that elapse from when the order happens to when the commodity arrives at the destination. Notice that customer's rating is proportional with $time$ and the lower the better. a and b are parameters to be determined later in section 7. I.e. f is the weighted average of $cost$ and $time$. $cost$ is with unit \$ and $time$ is with unit min.

  The constraint is that the scheme is represented by a simple path from $seller\_city$ to $purchaser\_city$. On the path, each 2 city is connected by exactly one transportation tool. $cost$ is calculated by simply add all the cost on each segment of the path.

  However, $time$ is hard to determine. Since the transportation tools has their $depart\_time$ and the commodity can't be transported immediately when it arrives. Therefore, $time$ can only be calculated by simulating the process. I.e. for each city u, if the commodity arrives earlier than the given transportation tool from u's $depart\_time$, add $time$ by $depart\_time - arrival\_time$. Else, it can only be transported out of city u next day. Hence, $time$ is added by $depart\_time - arrival\_time + 24 \times 60$.

- LP or ILP?

  We don't think this problem can be converted to LP or ILP. Since although the $cost$ function is linear, the $time$ function is not. As described above, $time$ function need to be calculated by simulation and it's discrete rather than continuous. Therefore, the objective function is not linear and we can't convert the problem to LP or ILP.

3.4 Complexity Analysis

This is a $NP\,Optimization$ problem $(l, sol, m, goal)$.

- l : $(G, orders)$. G is a given city network. $orders$ is a set of orders. This is poly-time recognizable.
- sol : $sol(G, orders)$ is a set of paths. For each $order \in orders$, there is a path from $seller\_city$ to $purchaser\_city$ in the set. Hence $|sol(x)| \leq p(|x|)$.
- m : $m(G, orders) = \sum_{order \in orders}(f(sol(order)))$. I.e. the sum of all the orders' schemes' overall evaluation. This function is poly-time computable.
- $goal$: minimize.

Therefore, this is a $NP\, Oprimation$ problem.

3.5 Algorithm Design

Although the time function is not linear and we can't find explicit weight function for edges, we can modify the classical $dijskra$ algorithm to generate a solution. The algorithm to solve one $order$ is shown in algorithm 1.

Algorithm 1: Schedule(G,$order$)

```
Input: city network $G$ and an order $order$.
Output: The optimal path from $order.seller\_city$ to $order.purchaser\_city$
1  $OPT\_cities \leftarrow \{seller\_city\}$;
2  Assign each city's $OPT\_value$ to be $\infty$;
3  $seller\_city.arrival \leftarrow 0$;
4  $seller\_city.cost \leftarrow 0$;
5  $seller\_city.OPT\_value \leftarrow 0$;
6  $current\_city \leftarrow seller\_city$;
7  while $OPT\_cities \neq V(G)$ do
8      foreach $city$ neighbor to $current\_city$ do
9          foreach $tool \in (current\_city, city).tools$ do
10             $time\_tmp \leftarrow$ the reach time of $city$ via $tool$;
11             $cost\_tmp \leftarrow current\_city.cost + tool.cost$;
12             $value\_tmp \leftarrow a \times cost\_tmp + b \times time\_tmp$;
13             if $value\_tmp < city.OPT\_value$ then
14                 $(city.arrival, city.cost, city.OPT\_value) \leftarrow$
                     $(time\_tmp, cost\_tmp, value\_tmp)$;
15                 $city.pre \leftarrow current\_city$;
16                 $city.tool \leftarrow tool$;
17     $current\_city \leftarrow$ the city in $cities/OPT\_cities$ with the lowest $OPT\_value$;
18     $OPT\_cities \leftarrow OPT\_cities \cup \{current\_city\}$;
19 $path \leftarrow$ the path recovered from $pre$ and $tool$;
20 return $path$;
```

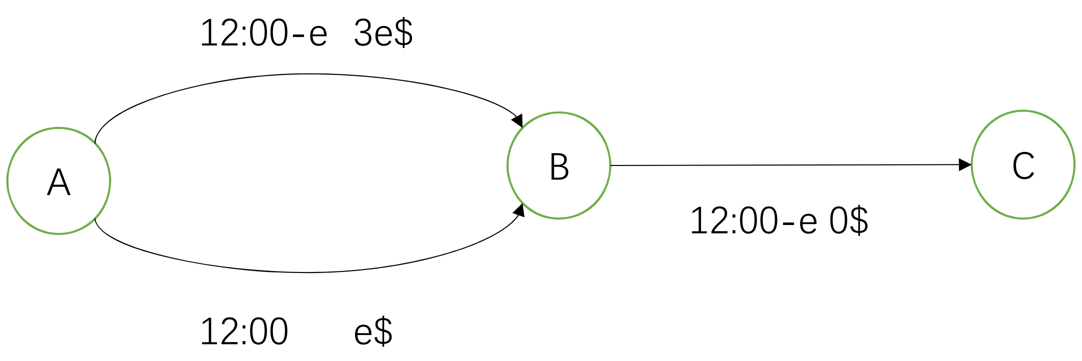

Figure 1: city network 1

3.6 Performance Analysis

1. Approximation Ratio

   Unfortunately, this algorithm doesn't have any approximation ratio theoretically. Consider the city network in figure 1. Suppose $a = b = 1$. $seller\_city = A$, $purchaser\_city = C$. The order happens at $12:00-e$. e's unit is min. The speed is infinite and delay is 0 ($time\_on\_way = 0$).

   $dijskra$ will choose the below edge to reach B from A, while the OPT chooses the above edge. $dijskra$ reaches C at $12:00-e$ the next day, using time $1440min$ and cost e$. OPT reaches C at $12:00-e$ this day, using time 0 and cost 3e$. Then we get

   $$\frac{dijskra}{OPT} = \frac{1440+e}{0+3e} = \frac{1440+e}{3e}$$

   If we choose e arbitrarily small, the ratio is arbitrarily large.

2. Approximation Difference

   Then we use another criterion approximation difference to evaluate the lower bound of the performance. Approximation difference is defined as $\max\{f(dijskra) - f(OPT)\}$. This criterion is more appropriate for this problem than approximation ratio. This is because users care about how much time the package is late for. And the company cares about how much money they waste. They care the difference rather than the ratio.

   Using this criterion, the algorithm is $poly-apx$. I.e. $\max\{f(dijskra) - f(OPT)\}$ is a polynomial function of the input. Denote the number of cities as k. The approximation difference

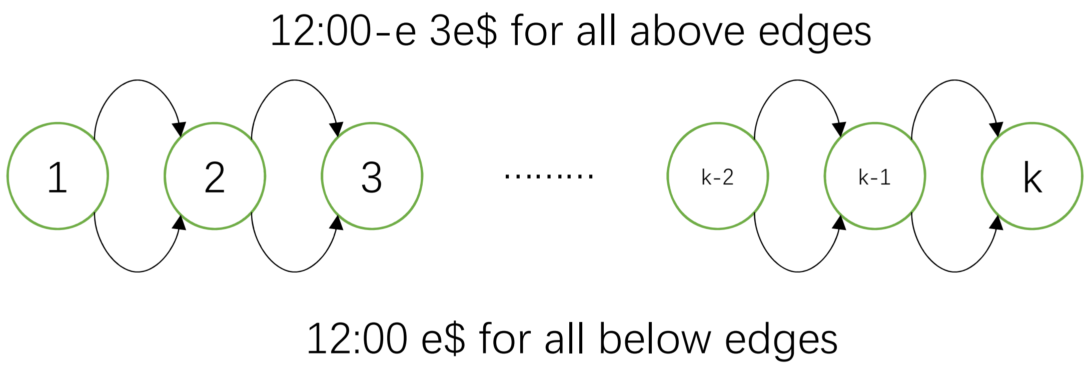

Figure 2: city network 2

is $1440b(k-1)$. Notice that the number of minutes in a day is 1440. Proof: if we don't care the waiting time, $dijskra$ is no worse than OPT. And the maximum time $dijskra$ waste on waiting is when it go through all k cities in a delivery and wait for about a day on each edge. Then the time wasted is $1440(k-1)$. This causes $1440b(k-1)$ difference to f.

This lower bound seems to be exaggerated. But it's actually a tight bound. See the network in figure 2. $time\_on\_way = 0$ for all edges and the order happens at $12:00-3$. OPT chooses the above edges for all, while $dijskra$ chooses the below ones. Then the $dijskra$ reaches k $k-1$ days later, while OPT reaches k immediately. If we choose e arbitrarily little, the approximation difference can be arbitrarily close to $1440(k-1)$.

3. Analysis in practice The $poly-apx$ seems to be terrible, but actually this algorithm works well in practice. This is because the real-world city network is a dense graph rather than the 1-dimension one shown in figure 2. And the probability that $dijskra$ chooses such a long path is nearly 0.

   In fact, we run the algorithm on the given city network. The largest f we get is

$$158885.46 = 18 \times 8676.70 + 1.2 \times 2254.05.$$

   We choose $a = 18$ and $b = 1.2$ in practice. This order costs 8760.70$ and uses about one day and a half. This is far lower than using $k-1$ days. Therefore, the algorithm works well in practice.

3.7 Efficiency Analysis

For each order, we run $dijskra$ once. Use $array$ for implementation. The algorithm checks each transportation tool at most once. Each check is $O(1)$. Also, each $edge$ and $vertex$ in G is checked at most once. Denote the total number of transportation tools in TableD-TransportationTools by $\#tools$. Denote the vertices set and edges set in G by V and E. Then the time complexity is $O(\#tools + |V| + |E|)$. Also, find the city with the lowest OPT_value is $O(|V|)$ in array implementation. Therefore, the time complexity is $O(\#tools + |V|^2)$. For each order, the algorithm runs once. Hence, the total time complexity is $O((\#tools + |V|^2) \times |orders|)$.

4 Problem 2

4.1 Problem Analysis

Assume the cost to construct a hub is $\hat{c}$.

$$\hat{c} = c_0 + c_1|orders|$$

c_0 is the cost for building the hub (a constant). $c_1|orders|$ is the cost for running the hub. This is proportional with the number of orders $|orders|$.

And assume the transportation cost between two hubs is $x (0 \le x \le 1)$ times that not between hubs. The f of a delivery is the sum of all the fs of its route.

For simplicity, **we assume that the delivery routes remain unchanged and only consider how to set up hubs and which transportation tool to use.** The idea is to sequentially check whether setting up a hub in a city can optimize $f(time, cost)$.

4.2 Problem Formulation and Complexity Analysis

We formulate this problem by $(I, sol, m, goal)$.

- $I : (G, orders, paths, \hat{c}, x)$. G is a city network. $orders$ is the set of orders. $paths$ is the scheme generated in problem 1. $paths$ is denoted by $|orders|$ paths in graph G. x and $\hat{c}$'s meaning are shown in section 4.1. This is poly-time recognizable.

- sol : $sol(G, orders, paths, \hat{c}, x)$ is the set of $(m\_paths, hubs)$. The requirements are:(i) the routes in m_paths is the same with that in $paths$ (transportation tools can be different); (ii) from one $hub \in hubs$ to one city, there is only one transportation tool used. This is also poly-time recognizable.

- m : The overall evaluation of scheme $paths$ is denoted by $sum\_f(paths)$. The overall evaluation of scheme m_paths is denoted by $sum\_f(m\_paths)$. Then $m(m\_paths, hubs) = sum\_f(paths) - sum\_f(m\_paths)$. I.e. the reduced amount of sum_f. This is poly-time calculatable.
- $goal$: maximize.

Therefore, we've shown that this is a NP-problem.

4.3 Algorithm Design

Since the orders and their routes have been solved in Problem 1, we have already known how many things of each type go through each city. We have also been restricted to using only one transportation tool from the hub to one city. So the transportation from the hub through different transportation tools has to be merged. Another check of which means of transportation is the best choice is therefore required. Algorithm 2 is the one checking whether construct a hub in city C. Algorithm 3 and 4 are 2 versions of the outer algorithms.

```
Algorithm 2: check($C$, $H$, $paths$)
Input: City $C$, current hub set $H$, current scheme $paths$;
Output: The benefit from building a hub in city $C$, new scheme $paths$;
1  //Notice that when calculating $sum\_f$, simulate the whole process, and take $H$ into
   consideration;
2  $benefit \leftarrow 0$;
3  foreach City $C'$ reachable from $C$ do
4      $max\_benefit \leftarrow 0$;
5      foreach $tool \in$ tools from $C$ to $C'$ do
6          $m\_paths \leftarrow paths$ but make all commodities from $C$ to $C'$ transported by $tool$;
7          if $sum\_f(paths) - sum\_f(m\_paths) > max\_benefit$ then
8              $max\_benefit \leftarrow sum\_f(paths) - sum\_f(m\_paths)$;
9              $opt\_paths \leftarrow m\_paths$;
10     $paths \leftarrow opt\_paths$;
11     $benefit += max\_benefit$;
12 return ($benefit - \hat{c}$,$paths$);
```

4.4 Performance Analysis

Notice that neither algorithm 2 nor 4 guarantees optimal solution. They are both sequential algorithms. Algorithm 2 sequentially determine the transportation tool from C to each city. Algorithm 4 sequentially determine whether constructing a hub in each city. However, the determination in previous iteration changes the scheme $paths$, thus influencing the later determinations. I.e. the order

Algorithm 3: $Set\_up\_hubs1(S, paths)$

```
Input: $S$ a set of cities, scheme $paths$ got in problem 1
Output: $H$ a set of hubs, new scheme $paths$;
1  $H \leftarrow \emptyset$;
2  for $c \in S$ do
3      if $check(c, H, paths).benefit > 0$ then
4          $H \leftarrow H \cup \{c\}$;
5          $paths \leftarrow check(c, H, paths).paths$;
6  return (H,paths);
```

Algorithm 4: $Set\_up\_hubs2(S, paths)$

```
Input: $S$ a set of cities,scheme $paths$ in problem 1;
Output: $H$ a set of hubs, new scheme $paths$;
1  $H \leftarrow \emptyset$;
2  while $H \neq S$ do
3      $max\_benefit \leftarrow 0$;
4      for $c \in S/H$ do
5          if $check(c, paths).benefit > max\_benefit$ then
6              $max\_benefit \leftarrow check(c, paths).benefit$;
7              $max\_city \leftarrow c$;
8      if $max\_benefit \leq 0$ then
9          break;
10     else
11         $H \leftarrow H \cup \{max\_city\}$;
12         $paths \leftarrow$ the scheme by setting $max\_city$ as a hub;
13 return (H,paths);
```

in which the cities are checked actually influence the outcome. Hence, the optimal solution is not guaranteed.

We gave 2 versions of the outer algorithm. Algorithm 3 simply checks each city and determine whether to make it a hub. This may lead to an arbitrarily bad result. E.g. it chooses the first city with very little benefit. However this choice makes all other cities unable to be hubs ($benefit \leq 0$). But the OPT can achieve much more benefit. To make this situation impossible, we also design another algorithm 4.

Algorithm 4 is poly-apx ($k - apx$). k is the number of cities. The benefit achieved by algorithm 4 is at least the benefit of constructing the first max_city. I.e. $max\_benefit$ in the first iteration. The benefit achieved by OPT is at most $k \times max\_benefit$. I.e. constructing hubs in each city and each with benefit the same as $max\_benefit$. Therefore, we get the following formula:

$$\frac{OPT.benefit}{sequential.benefit} < \frac{k \times max\_benefit}{max\_benefit} = k.$$

Hence, algorithm 4 is poly-apx.

4.5 Efficiency Analysis

The function $check$ is $O(\#tools_C \times |E| \times |orders|)$. $\#tools_C$ is the number of transportation tools from city C. $|E|$ is the number of edges in graph G. This is because that the algorithm simulates to get benefit $\#tools_C$ times. And each simulation requires $O(|E| \times |orders|)$ time.

For algorithm 3, the time complexity is $O(\#tools \times |E| \times |orders|)$. $\#tools$ is the total number of transportation tools. This is because adding up $\#tools_C$ for each city reaches $\#tools$.

For algorithm 4, the time complexity is $O(\#tools \times |E| \times |orders| \times |V|)$. $|V|$ is the number of vertices in G. This is because we check each city for at most $|V|$ times.

Since algorithm 4 is much slower, we implemented algorithm 3 instead.

We can improve the efficiency by only checking the orders whose delivery scheme has changed. However, this won't change the complexity in the worst case. This is because in the worst case, all orders go through all cities. Hence, when the transportation tools are merged, all orders' delivery schemes changed. Anyway, this optimization works well in practice.

5 Problem 3

5.1 Problem Analysis

For this problem we assume that the hub and the substation can function in the same city since hubs are capacitied. This is quite similar to problem 2 except that a little change has to be made to the algorithm to check for each city. Obviously, this is still a NP problem.

5.2 Algorithm Design

Since the hubs are capacitied, we have to take this into consideration and decide which commodities are to be transported through hubs and which through substations.

First, we should determine which orders to be delivered by the hub in the city. To maximize the benefit from the hub, we can treat the time and money we save as the value and the order's commodity's weight as the weight. Then convert the problem into a knapsack problem. The algorithm 5 illustrates our idea.

Algorithm 5: $checkTool(C, C', T)$

```
Input: Cities $C$ and $C'$ in which $C'$ is reachable from $C$ by $T$, a transportation tool;
Output: The maximum benefit of delivery from $C$ to $C'$ if we build a hub in city $C$ and use
        $T$ to deliver the goods;
for Commodity $c$ going through $C$ that are allowed in the hub of $C$ and in $T$ do
    $value[c] = f(p') - f(p)$ in which $p$ is the path with the old transportation while $p'$ goes
      by the new one;
    $r[c] = \frac{value[c]}{weight[c]}$;
$b \leftarrow$ The capacity of the hub;
Sort $r$ in non-increasing order $r_1, r_2, \ldots, r_k$;
$benefit \leftarrow 0$;
for $i \leftarrow 1$ to $n$ do
    if $b \geq weight[c_i]$ then
        $c \leftarrow \max_c\{value_{max}, r_i\}$;
    else
        $c \leftarrow \max_c\{value\}$;
        if $b < weight[c_i]$ then
            $c \leftarrow null$;
    Put $c$ into the hub;
    $benefit \leftarrow benefit + value[c]$;
    $b \leftarrow b - weight[c]$;
if $benefit$ < the benefit of putting only the feasible order with the largest $value$ then
    Only put the feasible order with the largest $value$ instead;
return $benefit$;
```

Then we can give the modified algorithm $check$ in algorithm 6

```
Algorithm 6: check($C$)
   Input: City $C$;
   Output: The overall benefits from building a hub in city $C$;
1  $benefit \leftarrow -\hat{c}$;
2  for City $C'$ reachable from $C$ do
3      $c \leftarrow \min_{T \in \text{Available tools from } C \text{ to } C'} checkTool(C, C', T)$;
4      $benefit \leftarrow benefit +$ (the current total f(time,cost) from$C$ to $C'$) $- c$;
5  return $benefit$;
```

And the outer algorithm remains unchanged.

Notice that when calculating the benefit, let those glass-made or inflammable products transported via the substation rather than hub in those cities which don't allow them. Also, the original scheme needs to be changed. For those orders with liquid and inflammable products, run the algorithm in problem 1 again, adding the constraint. Then run algorithms on the modified original scheme. Anyway, these 2 constraints will not cause substantial difference to our model.

5.3 Performance Analysis

The approximation ratio changes to $2k$, which is still poly-apx. This ratio is got by multiplying the ratio in problem 2 by 2. In class, we are taught that the knapsack problem is with approximation ratio 2. Hence multiply the approximation ratio in the 2 procedure and we got the overall approximation ratio.

5.4 Efficiency Analysis

Use the symbols defined in section 4.5. The time complexity of algorithm 5 is $O((|E|+\log|orders|)|orders|)$. Calculating each order's value is $O(|E||orders|)$. Sorting the orders is $O(|orders|\log|orders|)$.

The complexity changes because each check takes more time. The check function's complexity becomes $O(\#tools_C(|E|+\log|orders|)|orders|)$.

Then the complete algorithm's time complexity is $O(\#tools(|E|+\log|orders|)|orders|)$ (algorithm 3) or $O(\#tools(|E|+\log|orders|)|orders||V|)$ (algorithm 4).

6 Problem 4

6.1 Problem Analysis

Here is our assumption for this problem. If the $seller\_city$ is not a substation, it should be first delivered to a $substation$. As soon as the commodity reaches any substation, it should be delivered between substations, i.e. not going back to small cities. Once the commodity goes back to some small city, it should be transported to $purchaser\_city$ without going back to any substations. I.e. the commodities should be transported first between small cities, then substations and then again small cities. This is close to the reality.

Moreover, in reality those substations should be reachable from each other without small cities as intermediate. Otherwise, some substations may become islets. And any small city should be reachable from and to at least one substation. Otherwise, the city becomes an islet. Therefore, we make these 2 as our requirements for the data set.

We can revise our model by only set those cities with substations as the vertices. For those small cities without substations, associate them with the nearest substation. I.e. construct "huge vertices".

6.2 Complexity Analysis

Actually $problem_4 \leq_p problem_1$. Since for the situation where $seller\_city$ and $purchaser\_city$ are neither substation. All feasible delivery delivers from $seller\_city$ to substation s and then to substation t and finally to $purchaser\_city$. (s can be the same substation as t). Then we can let the algorithm exhaust all the possibilities of the choices of s and t. There are at most $O(|cities|^2)$ possibilities. Therefore, running the algorithms this many time can solve problem 4.

6.3 Modeling

- Modeling cities

  A city is modeled by a pair

$$(index, attribute).$$

  $index$ is the city's index. $attribute$ can be $substation$ or $small$.

- Modeling graph

  In this problem, we construct 2 graphs G and G_r. G is the same graph with problem 1. For G_r, delete all the cities without substation from the G in problem 1. And delete corresponding edges.

6.4 Algorithm Design

Although exhausting all probabilities can reach an optimized solution, it's too inefficient. Therefore, we revise the algorithm to be algorithm 7. Notice that theoretically this may lead to arbitrarily bad result. E.g. $seller\_city$ is not a substation but $purchaser\_city$ is. However the nearest substation from $seller\_city$ is infinitely far from the $purchaser\_city$. However, this can never happen in reality since the triangle law on real-world map. Therefore, this algorithm can lead to very close answer to the OPT in practice.

Algorithm 7: Schedule(G,$order$)

```
Input: city network $G$ and an order $order$.
Output: A path from $order.seller\_city$ to $order.purchaser\_city$
1  foreach $(u, v) \in E(G)$ do
2      Weight the edge by $Weight(u, v, G, order).minweight$;
3  foreach $(u, v) \in E(G_r)$ do
4      Weight the edge by $Weight(u, v, G_r, order).minweight$;
5  $G_{rr} \leftarrow$ graph $G$ deleting all substations;
6  $substation\_s \leftarrow order.seller\_city$;
7  $substation\_t \leftarrow order.purchaser\_city$;
8  if $order.seller\_city.attribute == substation$ &&
     $order.purchaser\_city.attribute == subtation$ then
9      Run Algorithm 1 in problem 1 on $G_r$;
10     return the path got;
11 if $order.seller\_city.attribute == small$ then
12     Use $dijskra$ on $G_{rr}$ to calculate the shortest path from $order.seller\_city$ to each small
         city;
13     foreach small city do
14         Update the shortest distance of neighbor substations;
15     Choose the substation $s_m$ with minimum weight of the shortest path;
16     $substation\_s \leftarrow s_m$;
17 if $order.purchaser\_city.attribute == small$ then
18     Use $dijskra$ on $G_{rr}$ to calculate the shortest path from each small city to
         $order.purchaser\_city$ ;
19     foreach small city do
20         Update the shortest distance of neighbor substations;
21     Choose the substation $t_m$ with minimum weight of the shortest path;
22     $substation\_t \leftarrow t_m$;
23 Run $dijskra$ algorithm $G_r$ to find a shortest path from $substation\_s$ to $substation\_t$;
24 Combine the 3 paths as $path$;
25 return $path$;
```

The algorithm's complexity is also

$$T(|orders|, \#cities) = |orders| \times \#cities^2.$$

7 Parameter Determination

In this section, we determine the parameters a and b defined in section 3.3. We want to reach the least $time$ and $cost$. To achieve this, we use $a \times cost + b \times time$ as an overall evaluation. And we need to carefully determine the 2 parameters a and b. Hence, we picked 2400 orders (not emergency) as a sample. Apply different parameters a and b on this sample and run the program. For each pair (a, b), calculate the average $time$ and $cost$ achieved. Then, use $time \times cost$ as the z-axis value and we get figure 3.

From this figure, we see that when $a = 18$, $b = 1.2$, the program achieve the lowest $time \times cost$. Hence, we determine $a = 18$ and $b = 1.2$ as the parameters used in the whole project.

For emergency orders, $time$ is more important. Therefore, we set b for those orders twice that for ordinary orders. I.e. $b = 2.4$ for emergency orders. Then the program care more on $time$ for those orders.

8 Performance Evaluation

Part of the program output. And analyze the trade-off between $time$ and $cost$.

9 Sensitivity Test

The effect of changing c_0, c_1, x in problem 2 is shown here. Also, $capacity$ in problem 3.

10 Summary

We refer to shortest path problem, sequential algorithm and knapsack problem to solve this project. And we sum the advantages and disadvantages of our model as follow:

Disadvantages:

1. In the worst case, the algorithms lead to poly-apx result in problem 2 and 3. And theoretically the algorithm can lead to arbitrarily bad result in problem 4. This may be far from the optimal. For problem 1, the "approximation difference" is also polynomial.

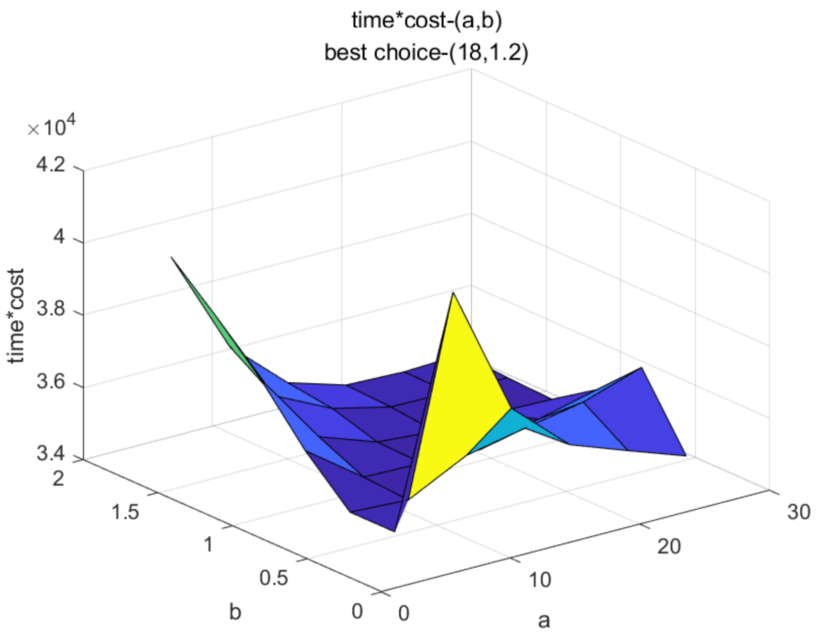

Figure 3: $time \times cost - (a, b)$

2. Although the algorithm is poly-time algorithm, it's still not efficient enough. Especially for problem 2 and 3, it simulates the whole process in the worst case for a check. This needs very much time.

Advantages:

1. The algorithm works out very close solution to OPT for problem 1 in practice. And it's efficient.

2. In practice, the algorithms performs well. Because of the triangle inequility of real-world maps, the algorithm for problem 4 works out schemes very close to the optimal ones. For problem 2 and 3, the worst case seldom happens in practice. Hence the efficiency and performance are much better.

3. All real-world objects (cities, transportation, $\cdots$) are all modeled to mathematical symbols. This helps the theoretical analysis and algorithm design.

11 Acknowledgement

From this project, we've learned very much. The largest achievement is that we've learned how to design algorithms in this project. And since we utilized the $dijskra$, sequential algorithm and $knapsack$ problem in this project, we become much more familiar to them. Also, as for the theoretical part, by proving the approximation ratio, we've got more familiar with the mathematical methods.

And from this project, we found that those problems in reality may seems simple. However, when we truly analyze them, investigate the algorithm, it turns out to be really hard. In problem 1, we find that the orders are independent with each other. And we thought we can solve the problem by processing the orders one by one. Then this becomes an easy problem. However, we struggled a lot to find such an algorithm and made many mistakes. And this problem even seems to be a NP complete one.

Thanks for professor Gao's instructions along the semester and the careful design of our labs and projects! We've really learned much from this course!

References

[1] Gao Xiaofeng. Slide12-ShortestPath. 2019

[2] Gao Xiaofeng. Slide15-NPReduction. 2019

[3] Gao Xiaofeng. Slide16-ApproximationI. 2019

[4] Gao Xiaofeng. Slide17-ApproximationII. 2019

Appendix