Package Delivery on "Double 11" Day

Project for Algorithm and Complexity

Kylin Chen (517030910155, k1017856853@icloud.com)
Fangyu Ding (517030910235, arthur_99@sjtu.edu.cn)
Hongzhou Liu (517030910214, deanlhz@sjtu.edu.cn)

Department of Computer Science,
Shanghai Jiao Tong University, Shanghai, China

Abstract. Package delivery is such a significant problem nowadays. The convenience and speediness of online shopping highly rely on an efficient and well-organized delivery network. Meanwhile, the delivery companies always want to reduce the cost of transportation while winning high rates from customers. The problem can be modelled as min-cost commodity flows over time. The characteristics of such kind of problem are networks with capacities and transit times. We also have other properties like the priority of orders of commodities, the ordering time and transportation restrictions with respect to special kinds of commodities. However, such kind of problems is really hard to solve. Though static *s*-*t* flow problem can be solved in polynomial time, the problem we encountered is almost NP-Hard with enormous input size. Thus, we came up with some approximations to get good feasible solutions, applied these algorithms in different scenes and compared their performance.
Keywords: Package Delivery, Network Flow, Routing, Flow over time.

0 Symbol Table

Define all the symbols that will be used later.

Table 1. Symbol Table

Symbol	Attribute
city	*index*, *kind*(*large*/*small*/*hub*), *capacity*
tool	*departure_city*, *arrival_city*, *time*, *average_delay*, *departure_time*, *unit_cost*, *type*
commodity	*index*, *unit_weight*, *type*
order	*seller_city*, *purchaser_city*, *order_time*, *commodity_index*, *amount*, *emergency*

1 Problem 1

1.1 Problem Analysis

In this part SF Express has its substations on all 656 cities covered in the orders. We came up with a network model. Regard *city* as vertex and *tool* as edge, we can construct a network and simulate the transportation of orders on it. Our cost function is defined as

$$C(p) = transport\_time^{rate} \times transport\_cost$$

Here p is the path that a particular order takes. And we evaluate the path using our cost function by the time the order takes from the time it was made to the time the package was sent to the consumer. We add an exponential *rate* to represent the weight of time in the cost function.
The problem is an NPO problem, we give the formal definition as follows:

- I: The network model $G = (V, E)$ and the set of *order*
- *sol*: A set of paths P representing the delivery scheme
- m: $m(G, order) = \sum_{p \in P} C(p)$
- *goal*: *min*

We consider that the problem is not a LP or ILP problem. Because the $transport\_time$ here is not linear. For example, there exist some orders that cannot be sent in the same day it was ordered. They will be delayed for several days thus the $transport\_time$ of a certain order will be a piecewise function. So we defined a non-linear object function, and constructed such network model.

1.2 Algorithm Design

Algorithm 1: *dfs*

Input: G, *source_city*, *target_city*, *order_time*, *arrival_time* , *visited*[], *path*, MIN, *optimal_path*, *depth_limit*

if *source_city* == *target_city and path's value* < *min* **then**
    MIN = *path*'s value;
    optimal_path = *path*;
    return;
end
if *depth_limit* == 0 **then**
    return;
end
visited[*source_city*] = *true*;
for *each* (*city*, *out_way*) *adjacent to source_city* **do**
    if *visited*[*city*] == *true and out_way.departure_time* ≥ *arrival_time* **then**
        visited[*city*] = *true*;
        path.push_back(*out_way*);
        // branch-cutting-off;
        if *path's value* < *min* **then**
            dfs(*G*, *city*, *target_city*, *order_time*, *out_way.arrival_time*, *visited*, *path*, *min*, *optimal_path*, *depth_limit*− 1);
        end
        path.pop_back();
        visited[*city*] = *false*;
    end
end

1.3 Theoretical Analysis

Complexity Analysis We implemented a depth-limited DFS algorithm which technically called **iterative deepening depth-first search(IDDFS)** to search for optimal solutions. The time complexity is $O(b^d)$, where b is the branching factor and d is the depth limit. With a depth limit, the nodes at depth d are expanded once, the nodes at depth $d-1$ are expanded twice. So the total number of expansions in an IDDFS is

$$b^d + 2b^{d-1} + 3b^{d-2} + ... + (d-1)b^2 + db + (d+1) = \Sigma_{i=0}^{d}(d+1-i)b^i$$

where b^d is the number of expansions at depth d, $2b^{d-1}$ is the number of expansions at depth $d-1$, and so on. Factoring out b^d gives

$$b^d(1 + 2b^{-1} + 3b^{-2} + ... + (d+1)^{-d})$$

Now let $x = \frac{1}{b}$, then we have

$$b^d(1 + 2x + 3x^2 + ...)$$

which converge to

$$b^d(1-x)^{-2}$$

for $abs(x) < 1$ Since $(1-x)^{-2}$ is a constant independent of d(the depth), if $b > 1$ (i.e., if the branching factor is greater than 1)

The time complexity is $O(b^d)$. Besides, as we also use the strategy of **branch-cutting-off**, when the total value of the current path IDDFS has found is larger than the current optimal, IDDFS would not carry on searching more cities at this branch.

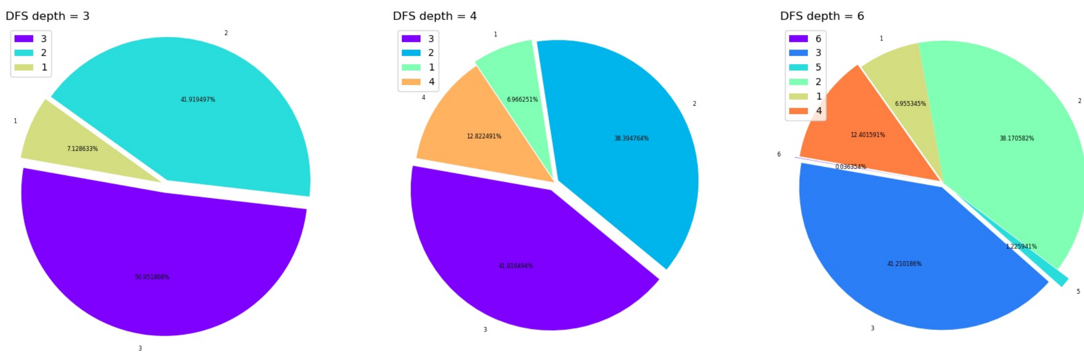
Therefore, the branching factor b would be divided by 2 because every time we want to carry on searching at one city, half of the out ways would be strictly **impossible to be optimal** in expectation. As a result, the factor b would be divided by 2 to be $O((\frac{b}{2})^d)$, which will significantly reduce the time complexity.

Efficiency Analysis The problem is an NP problem. We can find such a certifier that given a delivery scheme we can verify the correctness order-by-order. The procedure will take polynomial time and thus the problem is NP.

1.4 Performance Evaluation

Problem 1 is a foundation of our project, thus we take a detailed evaluation in this part.

First, we will take a look at the time of transfer for the optimal path of each order. Notice that $depth = \#transfer\_time + 1$ In our algorithm, we set a limitation of search depth. Once a certain order



(a) max depth = 3 (b) max depth = 4 (c) max depth = 6

Fig. 1. DFS depth

can find its optimal path under this limitation, we are sure it's an optimal solution. If not, we will let those orders who will not find it's optimal under the limitation, we will make it wait in some city and find a suboptimal solution. As we can see in these three graphs, with the increasing of limitation of depth, some part our pie chart decreased, which means some suboptimal solutions become optimal ones due to the increasing of the searching limitation. But increasing the limitation will bring us heavily computation cost and the worst circumstance will have limitation of 656, which is unaffordable to find all optimal solutions. However, we found that the proportion of orders which will take longer path to find their optimal is relatively small, thus we can take smaller depth limitation to make it an approximate algorithm. In the Problem 1-3, we choose the limitation as 4.

Second, we will discuss the choice of parameter $rate$ in our object function $C(p)$. We ran some testing programs and drew a graph to show the effect of our cost function on the decision of choosing paths in our algorithm. As we can see as long as the increasing of $rate$ the weight of time cost decreased sharply. It means that it will be hard to evaluate the cost if we choose a relatively large $rate$ for it will hide the contribution of time cost. So we choose $rate = \frac{1}{2}$ in our project to make a fair cost function to evaluate the performance.

In the third part, we analyze the approximation ratio. Notice that in our project we choose the depth of IDDFS as 4. So our approximate algorithm is IDDFS with depth of 4. However, it's really hard to compute the approximation ratio mathematically. We would rather compute the approximation ratio by comparing the result of our algorithm. As we mentioned before, the algorithm will surely find the global optimal if we set the depth as 656. It's because there're no negative loops in our network model and the depth of 656 implies that we will search all the possible paths.

2 Problem 2

2.1 Problem Analysis

In this section, we set some hubs in some cities. Hubs can gather packages and send them together to the same city with lower unit cost. However, the packages gathered together at a hub to one destination can

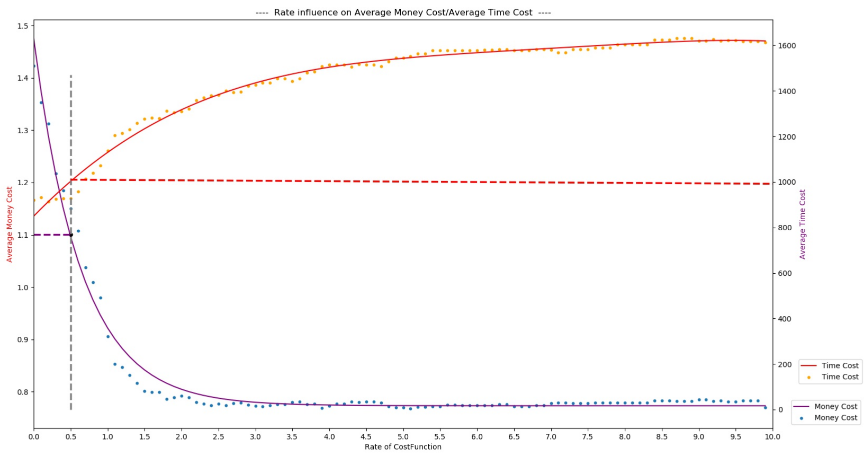


Fig. 2. Rate

only be sent by one transportation tool. We defined some new symbols
We consider it's an NPO problem as well.

Table 2. Problem 2 Symbols

Symbol	Definition
$\hat{C}$	The cost of setting a hub, a constant
$discount$	The discount rate on the cost when packages are sent from the same hub, $discount \in (0, 1)$
$\#hub$	The number of hubs
H	The set of cities where hubs are set

- I: The network model $G = (V, E)$, the set of $order$, the set of hubs H
- sol: A set of paths P representing the delivery scheme
- m: $m(G, order, H, \hat{C}, discount) = \sum_{p \in P} C(p)$.
- $goal$: min

2.2 Algorithm Design

First, we designed the algorithm to choose suitable cities to set hubs. Then, we designed the algorithm to find a new delivery scheme.

2.3 Theoretical Analysis

Complexity Analysis

Efficiency Analysis

2.4 Performance Evaluation

3 Problem 3

3.1 Problem Analysis

In real case, some other constraints should be considered: the hubs may be capacitated; some hubs may not accept some specific packages; some packages may not be transferred by some transportation tools. We listed our constraints for transportations as follows

Algorithm 2: dfs with hubs

Input: G, $out\_ways\_of\_hub$, $source\_city$, $target\_city$, $order\_time$, $arrival\_time$, $visited[]$, $path$, MIN, $optimal\_path$, $depth\_limit$

```
if source_city == target_city and path's value < min then
    MIN = path's value;
    optimal_path = path;
    return;
end
if depth_limit == 0 then
    return;
end
visited[source_city] = true;
if source_city is hub then
    for each (city, out_way) adjacent to source_city in Graph out_ways_of_hub do
        if visited[city] == true and out_way.departure_time ≥ arrival_time then
            visited[city] = true;
            path.push_back(out_way);
            if path's value < min then
                dfs(G, out_ways_of_hub, city, target_city, order_time, out_way.arrival_time, visited, path, min, optimal_path,
                 1);
            end
            path.pop_back();
            visited[city] = false;
        end
    end
end
else
    for each (city, out_way) adjacent to source_city in Graph G do
        if visited[city] == true and out_way.departure_time ≥ arrival_time then
            visited[city] = true;
            path.push_back(out_way);
            if path's value < min then
                dfs(G, city, target_city, order_time, out_way.arrival_time, visited, path, min, optimal_path, depth_limit−
                 1);
            end
            path.pop_back();
            visited[city] = false;
        end
    end
end
```

Table 3. Constraints on Transportations

Tool	Constraint
Trunk	None
Train	None
Plane	Inflammable Products, Liquid
Ship	Food

As for constraints on hubs, we randomly set some constraints on hubs. Here, the problem is still an NPO, we just add some new constraints to it. And we still cannot convert it to LP or ILP.

3.2 Algorithm Design

Algorithm 3: dfs with constraints

Input: G, $source\_city$, $target\_city$, $order\_time$, $arrival\_time$, $visited[]$, $path$, MIN, $optimal\_path$, $depth\_limit$, $commodity\_type$

```
if source_city == target_city and path's value < min then
    MIN = path's value;
    optimal_path = path;
    return;
end
if depth_limit == 0 then
    return;
end
visited[source_city] = true;
for each (city, out_way) adjacent to source_city do
    if visited[city] == true and out_way.departure_time ≥ arrival_time and out_way.type can deliver
      commodity_type then
        visited[city] = true;
        path.push_back(out_way);
        if path's value < min then
            dfs(G, city, target_city, order_time, out_way.arrival_time, visited, path, min, optimal_path, depth_limit−
              1, commodity_type);
        end
        path.pop_back();
        visited[city] = false;
    end
end
```

3.3 Theoretical Analysis

3.4 Performance Evaluation

4 Problem 4

4.1 Problem Analysis

In this problem, we suppose that the SF Express does not have substations in all cities, but only in big cities. Here we suppose that the big cities are those supporting airline service. This means that the SF Express should first take the packages from sellers to some substations, and when delivering the packages to purchasers, some substations should receive the packages first, and then send them to the city that the purchasers are in.

4.2 Algorithm Design

4.3 Theoretical Analysis

4.4 Performance Evaluation

5 First Section

5.1 A Subsection Sample

Please note that the first paragraph of a section or subsection is not indented. The first paragraph that follows a table, figure, equation etc. does not need an indent, either.

Subsequent paragraphs, however, are indented.

Algorithm 4: dfs for deliveries only happen between big cities

```
Input: G_among_big_cities, G_other_routes, source_city, target_city, order_time, arrival_time ,
       visited[], path, MIN, optimal_path, depth_limit_search_for_big, depth_limit_among_big,
       depth_limit_leave_from_big

if source_city == target_city and path's value < min then
    MIN = path's value;
    optimal_path = path;
    return;
end
//has not been in big cities (in the stage of searching for big cities);
if source_city is not big cities then
    //search for a big city;
    if depth_limit_search_for_big ne0 then
        visited[source_city] = true;
        for each (city, out_way) adjacent to source_city in Graph G_other_routes do
            if visited[city] == true and out_way.departure_time ≥ arrival_time then
                visited[city] = true;
                path.push_back(out_way);
                if path's value < min then
                    dfs(G_among_big_cities, G_other_routes, source_city, target_city, order_time,
                     arrival_time , visited[], path, MIN, optimal_path, depth_limit_search_for_big − 1,
                     depth_limit_among_big, depth_limit_leave_from_big);
                end
                path.pop_back();
                visited[city] = false;
            end
        end
    end
end
// has been in big cities;
else
    // we can go to another big city;
    if depth_limit_among_big ne0 then
        visited[source_city] = true;
        for each (city, out_way) adjacent to source_city in Graph G_among_big_cities do
            if visited[city] == true and out_way.departure_time ≥ arrival_time then
                visited[city] = true;
                path.push_back(out_way);
                if path's value < min then
                    dfs(G_among_big_cities, G_other_routes, source_city, target_city, order_time,
                     arrival_time , visited[], path, MIN, optimal_path, 0, depth_limit_among_big − 1,
                     depth_limit_leave_from_big);
                end
                path.pop_back();
                visited[city] = false;
            end
        end
    end
    // Also, we can leave from this big city to the destination;
    if depth_limit_leave_from_big ne0 then
        visited[source_city] = true;
        for each (city, out_way) adjacent to source_city in Graph G_other_routes do
            if visited[city] == true and out_way.departure_time ≥ arrival_time then
                visited[city] = true;
                path.push_back(out_way);
                if path's value < min then
                    dfs(G_among_big_cities, G_other_routes, source_city, target_city, order_time,
                     arrival_time , visited[], path, MIN, optimal_path, 0, 0,
                     depth_limit_leave_from_big − 1);
                end
                path.pop_back();
                visited[city] = false;
            end
        end
```

Sample Heading (Third Level) Only two levels of headings should be numbered. Lower level headings remain unnumbered; they are formatted as run-in headings.

Sample Heading (Fourth Level) The contribution should contain no more than four levels of headings. Table 4 gives a summary of all heading levels.

Table 4. Table captions should be placed above the tables.

Heading level	Example	Font size and style
Title (centered)	**Lecture Notes**	14 point, bold
1st-level heading	**1 Introduction**	12 point, bold
2nd-level heading	**2.1 Printing Area**	10 point, bold
3rd-level heading	**Run-in Heading in Bold.** Text follows	10 point, bold
4th-level heading	*Lowest Level Heading.* Text follows	10 point, italic

Displayed equations are centered and set on a separate line.

$$x + y = z \tag{1}$$

Please try to avoid rasterized images for line-art diagrams and schemas. Whenever possible, use vector graphics instead (see Fig. 3).

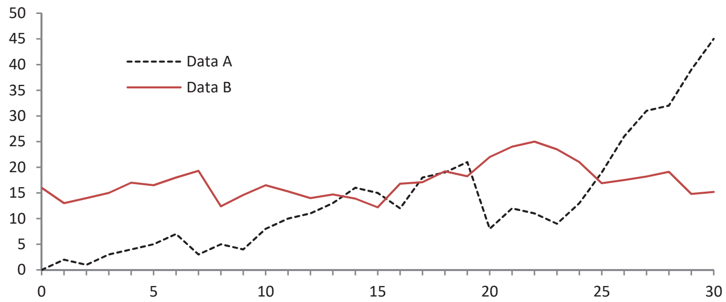


Fig. 3. A figure caption is always placed below the illustration. Please note that short captions are centered, while long ones are justified by the macro package automatically.

Theorem 1. *This is a sample theorem. The run-in heading is set in bold, while the following text appears in italics. Definitions, lemmas, propositions, and corollaries are styled the same way.*

Proof. Proofs, examples, and remarks have the initial word in italics, while the following text appears in normal font.

For citations of references, we prefer the use of square brackets and consecutive numbers. Citations using labels or the author/year convention are also acceptable. The following bibliography provides a sample reference list with entries for journal articles [1], an LNCS chapter [2], a book [3], proceedings without editors [4], and a homepage [5]. Multiple citations are grouped [1,2,3], [1,3,4,5].

Acknowledgements

Here is your acknowledgements. You may also include your feelings, suggestion, and comments in the acknowledgement section.

References

1. Author, F.: Article title. Journal **2**(5), 99–110 (2016)
2. Author, F., Author, S.: Title of a proceedings paper. In: Editor, F., Editor, S. (eds.) CONFERENCE 2016, LNCS, vol. 9999, pp. 1–13. Springer, Heidelberg (2016).
3. Author, F., Author, S., Author, T.: Book title. 2nd edn. Publisher, Location (1999)
4. Author, A.-B.: Contribution title. In: 9th International Proceedings on Proceedings, pp. 1–2. Publisher, Location (2010)
5. LNCS Homepage, http://www.springer.com/lncs. Last accessed 4 Oct 2017