Rapport de PIL

A solution to the static and deterministic vehicle routing problem

Nicolas Laurent-Brouty

CERMICS

Tuteur: Axel Parmentier
CERMICS
ENPC

Champs sur Marne
June 21, 2015
1 Introduction

1.1 Context

With the explosion of e-shopping over the last decade, the demand for fast shipping has increased. A lot of companies now offer delivery in 24 hours top, and have to handle an important number of packages in a limited amount of time. Moreover, there is a fierce competition between delivery companies to offer the cheapest costs to customers, which enable online shops to sell small packages. Thus, the companies have to adapt their delivery techniques, in order to get exploitation costs as low as possible.

Several companies have entered crowd-sourcing business, especially in the delivery market, such as Uber. They rely on the work of self-employed workers. This brings a new approach to the Vehicle Routing Problem. The Vehicle Routing Problem is an optimization problem which considers a fleet of vehicles and looks for the optimal set of routes to reach a given destination. Crowd-sourcing companies only pay for the route between the warehouse and the delivery location, so they have a different structure of operational costs. That is why it opens a new field of applications for optimization techniques. The object of this study will be to analyze methods of vehicle routing and adapt them to this type of demand. We will more particularly focus on the example of companies which offer delivery in the afternoon if you make your order before 3 pm, like Fnac Express in France.

1.2 Literature review

Our problem enters in a category of the Pickup and Delivery Problems. These problems can be considered as a class of Vehicle Routing Problem in which objects or people have to be transported between an origin and a destination. They can be divided in three categories (Berbeglia et al., 2010):

- many-to-many problems, in which there is no warehouse. Any location on a given map can serve as a source or as a destination for any commodity.

- one-to-many-to-one problems, in which commodities are initially available at the warehouse and destined to the customers. In addition, customers can send back commodities to the warehouse.

- one-to-one problems in which each commodity has a given origin and a given destination.

Berbeglia et al. (2010) present general solution concepts, such as basic solution strategies and algorithmic performance assessment. They also make a benchmark of the potential methods.

The problem of Pickup and Delivery can be either static or dynamic. In the first case, the requests are supposed to be known in advance as a deterministic process, which enables offline resolution (Berbeglia et al., 2007). On the other hand, the requests can be modelized by a stochastic process, which enables the implementation of online resolution (Pillac et al., 2013). Implementing online algorithms requires to adopt decision rules, such as waiting or relocation strategies (Saint-Guillain et al., 2015).
We will focus on an offline modelization, using basic heuristics, as described by Berbeglia et al. (2010). We will use Simulated Annealing Algorithm in the case of the Pickup and Delivery Problem (Gutenschwager et al., 2004).

1.3 Contributions
The contribution of this paper is threefold. First, we detail the taxonomy we developed to describe the problem. Second, we provide an analysis on the use of neighborhoods in the general case of Pickup and Delivery Problems. Third, we make a quantitative study on a simplified case which questions the interest of mutualization for delivery companies.

The paper is organized as follows. In section 2 we present the framework of our optimization problem. Section 3 and section 4 survey respectively the dominance-based algorithms and the heuristics for the any-warehouse problem. In section 5 we present the results of a numerical analysis of the any-warehouse without delivery windows problem.

2 Problem statement

2.1 General problem statement
Network and requests
Let $D = (V, A)$ be a directed graph (figure 1) modelling the road network of a city. $V$ is the set of vertices and $A$ is the set of arcs.

\begin{figure}
\centering
\begin{tikzpicture}
  \node (v1) at (0,0) [circle,draw] {$V_1$};
  \node (v2) at (2,0) [circle,draw] {$V_2$};
  \node (v3) at (1,-1.5) [circle,draw] {$V_3$};
  \node (v4) at (0,-3) [circle,draw] {$V_4$};

  \draw[->] (v1) edge node [right] {$A_1$} (v2);
  \draw[->] (v1) edge node [below] {$A_5$} (v4);
  \draw[->] (v2) edge node [right] {$A_2$} (v3);
  \draw[->] (v2) edge node [below] {$A_3$} (v4);
  \draw[->] (v3) edge node [left] {$A_4$} (v4);

\end{tikzpicture}
\caption{illustration of a basic directed graph}
\end{figure}

We consider the operations of a delivery company on one day. This company treats client requests. $R$ is the set of requests. The company receives $N$ requests for transportation. At first we constrain that the company treats and accepts all requests. Thus $N$ deliveries are realized.

The company owns $n$ warehouses. Let $W = (w_1, .., w_n)$ be the set of warehouses ($W \subset V$). We suppose that the company has an infinite number of vehicles, each with an infinite capacity of load.
In each warehouse the company realizes loading activities of goods into the vehicles.

The time horizon is denoted $H$. Each discrete time $t$ is in $[1, H]$.

Warehouse policy and feasible routes

We call route the path followed by a vehicle between a warehouse and its last destination. Mathematically, a route $\gamma$ is a sequence of operations $(o_1^\gamma, ..., o_n^\gamma)$. An operation is either a request or a loading. The set of operations is $O$. $\Gamma$ is the set of admissible routes. Each operation $o$ in $O$ is located on the vertex $v_o$ in $V$.

We distinct two different policies about the attribution of warehouses to the requests.

In the fixed-warehouse policy, each request specifies the provenance of each commodity. In this case a request $r \in R$ is denoted by $(v_r, w_r, t_r, b_r, e_r)$. The vertex $v_r \in V$ is the destination, the vertex $w_r$ is the location of the warehouse associated to $r$, $t_r \in T$ is the time at which the company receives the request, $b_r$ is the beginning time of the delivery window, and $e_r$ is the end time of the delivery window.

In the any-warehouse policy, the request does not include any specified warehouse. In this case a request $r \in R$ is fully described by $(v_r, t_r, b_r, e_r)$.

If a request $r$ has no constraint regarding the beginning of the delivery window, we set $b_r = 1$. If a request $r$ has no constraint regarding the end of the delivery window, we set $e_r = H$. We extend the notations defined in the case of requests to the loading operations. For each route $\gamma$, for each operation $o$ we write $o \in \gamma$ if and only if route $\gamma$ covers the operation $o$. Each request $r$ in $R$ can be served by only one route $\gamma$ in $\Gamma$.

For each $o_i$ in $\gamma$, $T_{o_i}^{\gamma, arr}$ is the time at which the delivery vehicle arrives at $v_{o_i}$. $T_{o_i}^{\gamma, dep}$ is the time at which the delivery vehicle leaves from $v_{o_i}$. For any $(v, w) \in V \pi_{u,v}$ is the shortest path, measured in time, between $v$ and $w$.

To be admissible, a route $\gamma$ must satisfy the following equations.

\[
T_{o_i}^{\gamma, arr} \leq T_{o_i}^{\gamma, dep} \leq \Pi_{o_i, o_{i+1}}
\]

Equation 1 means that the vehicle arrives at $v_i$ before leaving $v_i$. Equations 2 and 3 enforces the constraints of the delivery window.
Additionally, in the fixed-warehouse policy, to each request $r$ is associated a warehouse $w_r$. Then, an admissible route $\gamma$ must verify, for each $r \in \gamma$:

\begin{align*}
    t_r &\leq T_{w_r}^{\gamma,\text{dep}} \\
    T_{w_r}^{\gamma,\text{dep}} &\leq T_{v_r}^{\gamma,\text{arr}}
\end{align*}

(5) \quad (6)

In the any-warehouse policy, these conditions simplify to, for each $r \in \gamma$, there is a warehouse $w \in W$ such as:

\begin{align*}
    t_r &\leq T_{w}^{\gamma,\text{dep}} \\
    T_{w}^{\gamma,\text{dep}} &\leq T_{v_r}^{\gamma,\text{arr}}
\end{align*}

(7) \quad (8)

This implies that, for both policies, the path of each admissible route starts at a warehouse, i.e. $o_1^\gamma$ is a loading operation.

**Route cover**

Let $\Gamma$ be a set of $n_\Gamma$ routes which fulfill a set of requests $R$. Each request $r \in R$ must be served by one and only one route within the time window associated. This is equivalent to:

$$\exists! \gamma \in \Gamma : r \in \gamma, \forall r \in R$$

Crowd-sourcing delivery companies takes only into account the distance realized between warehouses and clients locations. We choose to measure the time needed by a fleet of vehicles to cover a certain set of requests. This measure represents an hourly rate, necessary for example for the salary of drivers.

For each $\gamma$ in $\Gamma$, $\Delta \gamma$ is the length, measured in time, of the route $\gamma$.

$$\Delta \gamma = T_{n_\gamma}^{\gamma,\text{arr}} - T_{0}^{\gamma,\text{dep}}$$

The cost of a solution $\Gamma$ is the sum of the lengths of its routes:

$$C(\Gamma) = \sum_{\gamma \in \Gamma} \Delta \gamma$$

**General problem statement**

The objective of the problem is to find a set of routes $\Gamma$ covering the set of request $R$ with a minimal cost $C(\Gamma)$.

**Several warehouses with delivery windows problem**

**Input.** A directed graph $D = (V, A)$. A set of $N$ requests $r \in R$. A set of warehouses: $W \subset V$.

**Output.** A feasible set $\Gamma$ of routes of minimum cost $C(\Gamma)$.

We note that this problem is NP-complete, which means that we cannot solve it exactly in a fixed amount of time.

**Remark 1.** There are in fact two distinct problems, depending on the delivery policy: the fixed-warehouse problem and the any-warehouse problem.
2.2 Variants of the problems

Table 1 details several subproblems of the Several Warehouses with Delivery Windows problem.

<table>
<thead>
<tr>
<th>denomination</th>
<th>specificity</th>
</tr>
</thead>
<tbody>
<tr>
<td>fixed-warehouse with delivery windows</td>
<td>((v_r, w_r, t_r, b_r, e_r))</td>
</tr>
<tr>
<td>fixed-warehouse without delivery windows</td>
<td>((v_r, w_r, t_r, b_r = 1, e_r = H))</td>
</tr>
<tr>
<td>any-warehouse with delivery windows</td>
<td>((v_r, t_r, b_r, e_r))</td>
</tr>
<tr>
<td>any-warehouse without delivery windows</td>
<td>((v_r, t_r, b_r = 1, e_r = H))</td>
</tr>
</tbody>
</table>

Table 1: Variants of the problem

2.3 Relations between variants of the problem

Each solution of the fixed-warehouse with delivery windows problem is solution to the fixed-warehouse without delivery windows problem. Each solution of the any-warehouse with delivery windows problem is solution to the any-warehouse without delivery windows problem. In addition, each solution of the fixed-warehouse problem is solution to the any-warehouse problem. As a consequence we have the following inequations.

\[
C(\text{fixed-warehouse without delivery windows}) \leq C(\text{fixed-arehouse with delivery windows})
\]
\[
C(\text{any-warehouse without delivery windows}) \leq C(\text{any-warehouse with delivery windows})
\]
\[
C(\text{any-warehouse}) \leq C(\text{fixed-arehouse})
\]

3 Dominance-based algorithms

3.1 Notion of domination between routes

In this section, we give several rules on feasible solutions.

In any given problem, a set of feasible routes \(S_2\) is said to be dominated by another set of feasible routes \(S_1\) if they cover the same vertices and \(C(S_1) < C(S_2)\).

**Proposition 1.** A set of feasible routes is an optimal solution to the problem if and only if it is non-dominated.

**Proof.** If a set of feasible routes is an optimal solution then its cost is the minimum cost of all feasible solutions and thus it is non-dominated. Conversely, if a set of feasible routes is non-dominated, then its cost is the minimum cost of all feasible solutions, which implies the optimality of this solution. \(\blacksquare\)
3.2 Dominated routes for the any-warehouse policy

**Proposition 2.** In the any-warehouse policy, a non-dominated route $\gamma$ has an unique warehouse located at the beginning of $\gamma$.

**Proof.** Let us consider a route $\gamma_1$ denoted by $(o_{\gamma_1}^1, \ldots, o_{\gamma_1}^n)$ and containing two warehouses. We already know that $v_{o_1} \in W$. Let $j \in [2, n_{\gamma_1}]$ such as $v_{o_j} \in W$. Let $\gamma_2$ be the route obtained by removing $o_j$ from $\gamma_1$. $C_{\gamma_2} = C_{\gamma_1} - C_{a_j} - C_{a_{j+1}} + C_{(j-1,j+1)}$. The shortest path verifies, for any $(u, v, w) \in V$, $\pi_{u, v} \leq \pi_{u, w} + \pi_{w, v}$ between to locations is the straight line. Then $C_{(j-1,j+1)} \leq C_{a_j} + C_{a_{j+1}}$ and $C_{\gamma_2} \leq C_{\gamma_1}$ and thus the route $\gamma_1$ is dominated. The same reasoning can be extended to routes covering three or more warehouses.

![Figure 2: illustration of the shortcut rule](image1)

On figure 2, the route does not need to cover the warehouse $w_2$ (as we consider vehicles with an infinite capacity, all requests can be loaded in $w_1$). Thus the route going straight from $r_2$ to $r_3$ dominates the road which covers $w_2$ between $r_2$ and $r_3$.

![Figure 3: illustration of the cut rule](image2)

On figure 3, the route $(w_1, r_1, r_2, w_2, r_3)$ is dominated by the set of routes $\{(w_1, r_1, r_2), (w_2, r_3)\}$. If a route contains several warehouses, we can either remove from it the warehouses (other than the first one) or cut it into a set of routes each containing one and only one warehouse.
This proposition reduces the number of feasible routes, and thus diminishes the size of any neighborhood. It improves the speed of optimization methods. Neighborhoods verifying this feasibility policy are obtained by removing from initial neighborhoods any route which does not satisfy proposition 2.

### 3.3 Dominated routes for the fixed-warehouse policy

**Proposition 3.** In the fixed-warehouse policy, a non-dominated route $\gamma$ goes through a given warehouse at most once.

**Proof.** Let us consider a route $\gamma_1$ denoted by $(o_1^{\gamma_1}, \ldots, o_n^{\gamma_1})$. We already know that $v_{o_1} \in W$. we suppose that there is $j \in [2, n^{\gamma_1}]$ such as $v_{o_j} = v_{o_1} \in W$. Let $\gamma_2$ be the route obtained by removing $o_j$ from $\gamma_1$. $C_{\gamma_2} = C_{\gamma_1} - C_{a_j} - C_{a_{j+1}} + C_{(j-1,j+1)} \leq C_{\gamma_1}$ and thus the route $\gamma_1$ is dominated. 

As proposition 2 in the any-warehouse policy, the proposition 3 enables to reduce the size of the neighborhoods used with heuristics methods and thus converge faster towards a given improving solution.

We note that the proposition 2 does not apply in the fixed-warehouse policy, as shown in figure 4. The set of routes $\{(w_1, r_1), (w_2, r_2)\}$ is dominated by the route $(w_1, w_2, r_2, r_1)$.

![Figure 4: illustration of the non-validity of proposition 2 in the fixed-warehouse policy](image-url)
4 Heuristics for the any-warehouse problem

A heuristic is an optimization algorithm designed to reach quickly good quality solutions, with no guarantee of optimality. A common class of heuristics is the class of neighborhood-based heuristics. The general idea of this class is to start from an initial solution and to descent in the neighborhood of the solution towards the nearest minimum (figure 5). In order to operate, we need to specify a type of neighborhood for each given solution. Mathematically, neighborhoods are a set of potential solutions and are obtained by operation transformations on the given solution.

![Cost](initial solution) → local descent → local minimum → global minimum

Figure 5: illustration of local descent mechanism

4.1 Local descent algorithm

On every neighborhood we can use an algorithm called *local descent*. We suppose that we want to obtain a solution with a cost inferior to maxCost.

```plaintext
start from an initial solution;
while Cost(solution) > maxCost do
    for each neighbor ∈ Neighborhood(solution) do
        if Cost(neighbor) < Cost(solution) then
            solution = neighbor;
        end
    end
end
```

**Algorithm 1**: Local descent algorithm

The difficulty with this algorithm is that its time of execution is proportional to the size of the neighborhood, which grows exponentially with the number of requests. Numerically, we verify that this algorithm is not viable as soon as we treat more than a hundred requests, which lead us to consider other techniques.
4.2 Simulated annealing algorithm

*Simulated annealing* is a randomized heuristic commonly used in optimization problems. The method was originally developed by Kirkpatrick (1984). Its inspiration comes from thermodynamic principles. Contrary to the local descent method, simulated annealing enables to move from a solution with a given cost to a solution with a higher cost, which may prevent from staying blocked in a local optimum (figure 6). The main force of simulated annealing, when compared to local descent, is that it converges significantly faster towards an improving solution, which is crucial when you optimize your solution in a finite amount of time. This is generally the case of most delivery companies, and the case of companies that want to implement fast delivery methods. But, as it does not sweep all neighborhoods of a given solution, it may not reach the optimal solution.

![Figure 6: illustration of simulated annealing mechanism](image)

We implement a basic simulated annealing algorithm. For details of the behavior see the work of Rutenbar *et al.* (1989). We impose $t_{max}$ as the number of seconds during which the algorithm will run and optimize the solution.
**Data**: a given Neighborhood policy; $\alpha < 1$; $T_{initiale} > 0$; start from an initial solution; initial time $= 0$;

**while** time $< t_{max}$ **do**

- neighbor = solution $\rightarrow$ goToNeighbor; nombreAleatoire = random([0,1]); temperature = $T_{initiale}$;
- if Cost(neighbor) $<$ Cost(solution) **or** nombreAleatoire $<$ $e^{-(\text{Cost(neighbor)}-\text{Cost(solution)})/\text{temperature}}$ **then**
  - solution = neighbor;
- end
- temperature = $\alpha$.temperature;

**end**

return solution ;

**Algorithm 2**: Simulated annealing algorithm

$\alpha$ represents a cooling coefficient. In the beginning of the optimization process, the algorithm can easily get out of a local minimum, to converge quickly towards the local minimum. At each step, we diminish the temperature with the cooling coefficient to limitate the capacity of the algorithm to get out of a local minimum as it converges towards the global minimum.

### 4.3 Neighborhoods for local heuristics

We consider a set of feasible routes $\Gamma = (\gamma_1, \ldots, \gamma_k)$. We can identify several neighborhoods of $\Gamma$. They are generated by applying transformations on $\Gamma$.

**Reordering a route**

The sequence of a given route $\gamma \in \Gamma$ can be rearranged in order to minimize the cost of the route. It brings the following optimization problem, which is similar to the *travelling salesman problem*:

**Ordering a route problem**

**Input.** A set of $n$ operations $o \in O$ containing only one loading operation.

**Output.** A sequence $(o_1, \ldots, o_n)$ with minimal cost $C$ ($o_1$ is the loading operation).

**Add-remove/swap**

The add-remove process consists in attributing a given operation initially in $\gamma_i$ to a different route $\gamma_j$. The swap process consists in exchanging two operations from two different routes, meaning that $(o_k^{\gamma_j}, o_l^{\gamma_j})$ is replaced by $(o_k^{\gamma_i}, o_l^{\gamma_i})$.

**Merger of routes**

A set of two routes $\{(o_1^{\gamma_1}, \ldots, o_k^{\gamma_1}), (o_1^{\gamma_2}, \ldots, o_l^{\gamma_2})\}$ is merged to obtain a single route $(o_1^{\gamma_1}, \ldots, o_k^{\gamma_1}, o_1^{\gamma_2}, \ldots, o_l^{\gamma_2})$ and then removing from it the unnecessary loading operations, in accordance with proposition 2 and proposition 3.
Splitting of a route

We split a route \((o_1^{\gamma_1},...,o_k^{\gamma_k})\) into two routes \(\{(o_1^{\gamma_2},...,o_l^{\gamma_2}), (w,o_{l+1}^{\gamma_1},...,o_k^{\gamma_1})\}\), where \(w \in W\) (preferably with \(w\) the closest warehouse of \(o_{l+1}\) to minimize the cost).

Uncrossing routes

If a set of routes \(S\) includes two routes \(\gamma_1\) and \(\gamma_2\) such as their path cross each other, we create a new set of route \(S'\) obtained from \(S\) by uncrossing \(\gamma_1\) and \(\gamma_2\). If \(D\) is endowed in \(\mathbb{R}^2\) equipped with the usual euclidian norm, considering \((u,v,w,x) \in V\), the routes \((u,v)\) and \((w,x)\) cross each other if their respective distances satisfies the following equations (see Figure 7).

\[
\begin{align*}
\quad &d_{(u,v)} \geq d_{(u,x)} \text{ or } d_{(u,v)} \geq d_{(u,w)} \\
\quad &d_{(w,x)} \geq d_{(w,u)} \text{ or } d_{(w,x)} \geq d_{(w,v)}
\end{align*}
\]

![Figure 7: how to identify two crossing paths](image)

4.4 Initialization algorithm

Each problem has the same input: a set \(R\) of \(N\) requests which must be delivered by the company. Additionally, we have a set \(W\) of warehouses available. Each optimization algorithm detailed requires an initial solution to start. Thus, we have to create a feasible solution. We propose the following pseudo-algorithm:

```plaintext
for each \(r \in R\) do
    find the closest warehouse \(w \in W\);
    create the road \((w,r)\);
    insert the road into the feasible solution;
end
```

**Algorithm 3:** Initialization Algorithm

This algorithm simulates the delivery policy of delivery companies which do not mutualize orders, as it affects to each request the closest warehouse.
5 Numerical analysis of the Any-Warehouse without Delivery Windows problem

5.1 Main results

We chose to study the any-warehouse without delivery windows problem, which represents the case of a "classical" delivery company which aims at reducing its delivery costs. The objective is to analyze the impact of mutualization on delivery costs.

We present in table 2 a summary of all instances. We ran a significant number of simulations, while modifying the different parameters of the simulation:

- the number of requests,
- the number of warehouses,
- the size of the graph,
- the duration of optimization $t_{max}$,
- the initial value of thermodynamic temperature $T_{initiale}$,
- the cooling coefficient $\alpha$.

During the tests, we used two distinct graphs. The first contained 9 vertices and 24 arcs, while the second contained 100 vertices and 360 arcs. We implemented the simulated annealing algorithm, and considered the add-remove neighborhood. The simulations were made on a personal computer equipped with a dual core processor and 8 GB of RAM.

We note that the ratios of cost and $n\text{(routes)}$, computed respectively as the cost of the initial solution over the cost of the optimized solution and as the initial number of routes over the optimized number of routes, decreases with the apparent complexity of the initial problem. We consider that the apparent complexity of the input increases with the number of requests, the number of warehouses and the size of the graph. On the small graph, the algorithm shows good optimization results, even though its global performances decreases when facing more than 10 000 requests. On the large graph, its performances plunge, with quickly less than 10 % improvement.

We remark that the optimization algorithm is efficient on the small graph, and with a small number of warehouses. In this case, the mutualization of deliveries is a significant improvement, and enables a delivery company to significantly reduce its costs. In the case of complex problems, 100 seconds of optimization with our algorithm is not sufficient to diminish significantly the costs. The number of potential combinations increases exponentially with the number of vertices, arcs and warehouses. Thus an optimization process needs more time to sweep a neighborhood. A delivery company in that case may choose to run optimization processes for several minutes or hours or choose not to mutualize deliveries.
<table>
<thead>
<tr>
<th>instance</th>
<th>graph</th>
<th>size of</th>
<th>cost of initial solution</th>
<th>improved solution</th>
<th>ratios</th>
</tr>
</thead>
<tbody>
<tr>
<td>W</td>
<td>R</td>
<td>min cost</td>
<td>n(routes)</td>
<td>cost</td>
<td>n(routes)</td>
</tr>
<tr>
<td>1</td>
<td>9 v 24 a</td>
<td>11 1784 6479</td>
<td>598</td>
<td>125</td>
<td>10.83</td>
</tr>
<tr>
<td>2</td>
<td>9 v 24 a</td>
<td>11 1784 6479</td>
<td>590</td>
<td>124</td>
<td>10.98</td>
</tr>
<tr>
<td>3</td>
<td>9 v 24 a</td>
<td>11 1784 6479</td>
<td>590</td>
<td>124</td>
<td>10.98</td>
</tr>
<tr>
<td>4</td>
<td>9 v 24 a</td>
<td>11 1784 6479</td>
<td>590</td>
<td>124</td>
<td>10.98</td>
</tr>
<tr>
<td>5</td>
<td>9 v 24 a</td>
<td>11 1784 6479</td>
<td>588</td>
<td>124</td>
<td>11.02</td>
</tr>
<tr>
<td>6</td>
<td>9 v 24 a</td>
<td>11 1784 6479</td>
<td>590</td>
<td>124</td>
<td>10.98</td>
</tr>
<tr>
<td>7</td>
<td>9 v 24 a</td>
<td>11 1784 6479</td>
<td>590</td>
<td>124</td>
<td>10.98</td>
</tr>
<tr>
<td>8</td>
<td>9 v 24 a</td>
<td>11 1784 6479</td>
<td>590</td>
<td>124</td>
<td>10.98</td>
</tr>
<tr>
<td>9</td>
<td>9 v 24 a</td>
<td>11 1784 6479</td>
<td>590</td>
<td>124</td>
<td>10.98</td>
</tr>
<tr>
<td>10</td>
<td>9 v 24 a</td>
<td>11 1784 6479</td>
<td>590</td>
<td>124</td>
<td>10.98</td>
</tr>
<tr>
<td>11</td>
<td>9 v 24 a</td>
<td>11 1784 6479</td>
<td>590</td>
<td>125</td>
<td>10.98</td>
</tr>
<tr>
<td>12</td>
<td>9 v 24 a</td>
<td>11 3554 12869</td>
<td>1862</td>
<td>448</td>
<td>6.91</td>
</tr>
<tr>
<td>13</td>
<td>9 v 24 a</td>
<td>11 5379 19588</td>
<td>4511</td>
<td>1105</td>
<td>4.34</td>
</tr>
<tr>
<td>14</td>
<td>9 v 24 a</td>
<td>11 7092 25355</td>
<td>9270</td>
<td>2430</td>
<td>2.74</td>
</tr>
<tr>
<td>15</td>
<td>9 v 24 a</td>
<td>11 8792 31192</td>
<td>15975</td>
<td>4230</td>
<td>1.95</td>
</tr>
<tr>
<td>16</td>
<td>9 v 24 a</td>
<td>11 10510 37149</td>
<td>23078</td>
<td>6167</td>
<td>1.61</td>
</tr>
<tr>
<td>17</td>
<td>9 v 24 a</td>
<td>11 12276 43290</td>
<td>30430</td>
<td>8273</td>
<td>1.42</td>
</tr>
<tr>
<td>18</td>
<td>9 v 24 a</td>
<td>11 13990 49438</td>
<td>37947</td>
<td>10470</td>
<td>1.3</td>
</tr>
<tr>
<td>19</td>
<td>9 v 24 a</td>
<td>11 15735 55533</td>
<td>44587</td>
<td>12320</td>
<td>1.25</td>
</tr>
<tr>
<td>20</td>
<td>9 v 24 a</td>
<td>11 17482 61623</td>
<td>51937</td>
<td>14396</td>
<td>1.19</td>
</tr>
<tr>
<td>21</td>
<td>9 v 24 a</td>
<td>11 19228 67813</td>
<td>59275</td>
<td>16375</td>
<td>1.02</td>
</tr>
<tr>
<td>22</td>
<td>9 v 24 a</td>
<td>11 21074 74009</td>
<td>67875</td>
<td>20475</td>
<td>1.01</td>
</tr>
<tr>
<td>23</td>
<td>9 v 24 a</td>
<td>11 22920 80205</td>
<td>75375</td>
<td>24475</td>
<td>1.01</td>
</tr>
<tr>
<td>24</td>
<td>9 v 24 a</td>
<td>11 24766 86401</td>
<td>82905</td>
<td>28475</td>
<td>1.01</td>
</tr>
<tr>
<td>25</td>
<td>9 v 24 a</td>
<td>11 26612 92597</td>
<td>90405</td>
<td>32475</td>
<td>1.01</td>
</tr>
<tr>
<td>26</td>
<td>9 v 24 a</td>
<td>11 28458 98793</td>
<td>97905</td>
<td>36475</td>
<td>1.01</td>
</tr>
<tr>
<td>27</td>
<td>9 v 24 a</td>
<td>11 30304 105179</td>
<td>105405</td>
<td>40475</td>
<td>1.01</td>
</tr>
<tr>
<td>28</td>
<td>9 v 24 a</td>
<td>11 32150 111441</td>
<td>111905</td>
<td>44475</td>
<td>1.01</td>
</tr>
<tr>
<td>29</td>
<td>9 v 24 a</td>
<td>11 33996 117757</td>
<td>118405</td>
<td>48475</td>
<td>1.01</td>
</tr>
<tr>
<td>30</td>
<td>9 v 24 a</td>
<td>11 35842 124053</td>
<td>124905</td>
<td>52475</td>
<td>1.01</td>
</tr>
<tr>
<td>31</td>
<td>9 v 24 a</td>
<td>11 37688 130369</td>
<td>131405</td>
<td>56475</td>
<td>1.01</td>
</tr>
<tr>
<td>32</td>
<td>9 v 24 a</td>
<td>11 39534 136681</td>
<td>137905</td>
<td>60475</td>
<td>1.01</td>
</tr>
<tr>
<td>33</td>
<td>9 v 24 a</td>
<td>11 41380 142997</td>
<td>144405</td>
<td>64475</td>
<td>1.01</td>
</tr>
<tr>
<td>34</td>
<td>9 v 24 a</td>
<td>11 43226 149313</td>
<td>150905</td>
<td>68475</td>
<td>1.01</td>
</tr>
<tr>
<td>35</td>
<td>9 v 24 a</td>
<td>11 45072 155629</td>
<td>157405</td>
<td>72475</td>
<td>1.01</td>
</tr>
<tr>
<td>36</td>
<td>9 v 24 a</td>
<td>11 46918 161945</td>
<td>163905</td>
<td>76475</td>
<td>1.01</td>
</tr>
<tr>
<td>37</td>
<td>9 v 24 a</td>
<td>11 48764 168261</td>
<td>169405</td>
<td>80475</td>
<td>1.01</td>
</tr>
<tr>
<td>38</td>
<td>9 v 24 a</td>
<td>11 50610 174577</td>
<td>176905</td>
<td>84475</td>
<td>1.01</td>
</tr>
<tr>
<td>39</td>
<td>9 v 24 a</td>
<td>11 52456 180893</td>
<td>183405</td>
<td>88475</td>
<td>1.01</td>
</tr>
<tr>
<td>40</td>
<td>9 v 24 a</td>
<td>11 54302 187219</td>
<td>186905</td>
<td>92475</td>
<td>1.01</td>
</tr>
</tbody>
</table>

Table 2: Summary of instances realized
5.2 Impact of time on algorithmic performance

First, we assessed the impact of $t_{max}$ on the cost of the optimized solution. The constant parameters for all instances were chosen as follows: $T_{initiale} = 1$, $\alpha = 0.95$. We realized for each instance simulations for $t_{max}$ varying from 1 to 100 seconds. We obtained the following results (figure 8).

![Figure 8: impact of $t_{max}$ on the cost of the optimized solution](image)

The bigger the time of optimization the bigger the number of iterations realized by the simulated annealing algorithm, which justifies that the cost decreases with the duration of the optimization method. 100 seconds of optimization provide a significant optimization of the initial solution on the small graph. Nevertheless, we note that 100 seconds are not sufficient on the large graph to significantly improve the initial solution. Our algorithm does not converge quickly enough in this case.
5.3 Impact of initial temperature on algorithmic performance

We assessed the impact of $T_{\text{initial}}$ on the cost of the optimized solution. The constant parameters for all instances were chosen as follows: $t_{\text{max}} = 1$ and $t_{\text{max}} = 10$, $\alpha = 0.95$. We realized for each instance simulations for $T_{\text{initial}}$ varying from 0.1 to 10. We obtained the following results (figure 9 for $t_{\text{max}} = 1$, figure 10 for $t_{\text{max}} = 10$).

![Figure 9: impact of $T_{\text{initial}}$ with $t_{\text{max}} = 1$ on the cost of the optimized solution](image)

The value of $T_{\text{initial}}$ does not seem to have a significant impact in the optimization process, with the chosen graph, warehouses, requests and numerical parameters. This might be explained by the fact that the considered neighborhoods do not have many neighborhoods, and that its behavior is similar to local descent algorithm.
5.4 Impact of cooling parameter on algorithmic performance

We assessed the impact of the cooling parameter $\alpha$ on the cost of the optimized solution. The constant parameters for all instances were chosen as follows: $t_{\text{max}} = 1$ and $t_{\text{max}} = 10$, $T_{\text{initiale}} = 1$. We realized for each instance simulations for $\alpha$ varying from 0.1 to 1. We obtained the following results (figure 11 for $t_{\text{max}} = 1$, figure 12 for $t_{\text{max}} = 10$).

We note that, as for $T_{\text{initiale}}$, the impact of the cooling parameter on algorithmic performances is not significant. The cost of the optimized solution does not vary noticeably with the evolution of $\alpha$. In the case of $t_{\text{max}} = 1$, we do not see any relation between the value of $\alpha$ and the final cost. With a process ten times longer (figure 12) the impact of $\alpha$ is more significant, as we notice a diminution of the final cost if $\alpha \geq 0.6$. The cooling is slower when $\alpha$ is close to 1, which means that the process can more easily leave local minimums.
Figure 11: impact of $\alpha$ with $t_{\text{max}} = 1$ on the cost of the optimized solution

Figure 12: impact of $\alpha$ with $t_{\text{max}} = 10$ on the cost of the optimized solution
References


