Master of Science in Informatics at Grenoble
Master Mathématiques Informatique - spécialité Informatique
option ROCO

Exact algorithms for picking problem

Lucie Pansart

June 2016

Research project performed at G-SCOP

Under the supervision of:
Dr. Nicolas Catusse
Dr. Hadrien Cambazard

Defended before a jury composed of:
Dr. Gülgün Alpan
Dr. Nadia Brauner
Dr. Massih-Reza Amini
Dr. Catherine Berrut
Dr. Jérôme Euzenat
Dr. Eric Gaussier
Dr. Marie-Laure Espinousse
Dr. Gautier Stauffer

June 2016

Abstract

This paper considers the problem of order picking in warehouse. Order picking is a major issue in the supply-chain because of its cost in time and labor force. Each sub-optimal choice can be really expensive for warehousing professionals. This leads to the necessity of getting exact algorithms to find the optimal path in the warehouse, and thus, collect all the orders in the minimal time. For the generic case of a rectangular and regular warehouse with any number of aisles and cross-aisles, the problem is $\mathcal{NP}$-hard.
In this report, we present a mixed-integer programming approach to solve this problem and show that, with improvements like preprocessing and cutting planes, this approach is effective. Performance are compared between other exact approaches for this problem, including a dynamic programming and the TSP solver Concorde.

Acknowledgments

I take the opportunity to thank those who make this work possible.

First, it's important to me to thank my two directors Hadrien Cambazard and Nicolas Catusse. They proposed me this project and adapted it to fit my wishes. I had a big liberty and an important support all along my internship. I was free to investigate the tracks that interested me but they were available to answer my questions and to help me whenever I needed it.

I would like to thank Gautier Stauffer and Marie-Laure Espinousse, in charge of the master ROCO. None of this would have been possible if they didn't make all the efforts to obtain the opening of the master despite the financial difficulties. Their engagement enabled our success.

Finally, I want to thank Nadia Brauner who guided me and advised me honestly until now.

Contents

1 Introduction

Throughout industrial history, companies have always been seeking to reduce their costs by optimizing the logistical costs. Recently, the financial crisis as well as the transformation of the way of buying weigh upon the logistical processes: distribution centers are centralized, customer orders are smaller and more frequent, every step of the supply chain must be responsive and flexible. To gain productivity and competitiveness, every warehouse process must be optimized.
At the heart of the logistic, the order picking activity consists in collecting products from storage in a specific quantity given by a customer order. This process is often considered as the most important warehousing process.

An efficient way to optimize order picking, when it's performed by an order picker moving in a warehouse, is to reduce its travel time. Thus, we are concerned with the following issue: **how to optimize the routing time in the warehouse ?**
This problem is called picker routing problem, or picking problem through misuse of language.

This problem is $\mathcal{NP}$-hard. Indeed, it relates to the Traveling Salesman Problem in a grid graph (shown $\mathcal{NP}$-hard in 1982 [24]). Thus, it is likely that no polynomial time algorithm exists to solve it optimally. But it is possible to create exact approaches with reasonable computing time. We present here a complete algorithm which proves to be efficient on an academic benchmark.

This report begins with a description of the picking problem in section 2, a literature review (section 3) and a presentation of some motivations for this problem in section 4.
This is followed by two ways of modeling the problem, presented in sections 5 and 6 and improvements on these models (section 7). Finally, we analyze and compare different mixed-integer linear programs (MILP) of each model in section 8 before presenting experimental results (section 9).

2 Problem statement

The problem is stated as follows: given n products to pick in a rectangular warehouse, what is the shortest tour (beginning and ending at the depot) to collect all these products ?
It is a particular case of the Traveling Salesman Problem (TSP) where the salesman is the order picker and towns are products to collect.

2.1 Description of an instance

A picking problem is defined by a warehouse and a picking list. The picking list is a set of n products described by their location in the warehouse.

A warehouse is made of narrow vertical aisles and horizontal cross-aisles. Products are located on both sides of narrow aisles. Each side of a narrow aisle is referred to as a half-aisle. Cross-aisles do not contain any products but enable the order picker to navigate in the warehouse.

Hypothesis 1. *Aisle are narrow enough so that the cost of crossing an aisle is null.*

This hypothesis implies that half-aisles are not used in mathematical purpose but only to represent data.

Hypothesis 2. *All the aisle lengths are the same and all cross-aisles lengths are the same.*

Following [39], the parameters describing the layout and dimensions of a rectangular warehouse are summarized in Figure 1 and contain in particular:

- `warehouseX`: table where `warehouseX[i]` gives the half-aisle index of order i.
- `warehouseY`: table where `warehouseY[i]` gives the location index of order i.
- `locationsX`: table of size $n+1$ containing the **abscissa** of the depot and orders
- `locationsY`: table of size $n+1$ containing the **ordinate** of the depot and orders
- `distances`: symmetric matrix of size $n+1$ containing minimal distances between two orders or between an order and the depot.

We introduce some useful notations:

- A **block** contains all locations between two cross-aisles.
- A **sub-aisle** contains all items of one aisle in one block and the two intersections surrounding them.

Figure 1 shows an example of a warehouse description. An example of an instance file is given in appendix A.

Remark. The data `distance`, `locations` and `warehouse` are redundant.

2.2 Some definitions

Graph theory We use the usual following notations of graph theory and refer by $G = (V, A)$ to a directed graph with a set V of vertices and a set A of arcs. The neighbors of a vertex i are denoted by Γ:

$\Gamma^+\{i\} = \{j \in V \vert (i,j) \in A\} \subset V$	is the set of "outgoing" **neighbors**
$\Gamma^-\{i\} = \{j \in V \vert (j,i) \in A\} \subset V$	is the set of "incoming" **neighbors**
$\Gamma\{i\} = \Gamma^-\{i\} \cup \Gamma^+\{i\}$	is the set of **neighbors**

The arcs connected to vertex i are denoted by δ:

$\delta^+\{i\} = \{(ij) \in A \vert \forall j \in V\} \subset A$	is the set of "outgoing" **arcs**
$\delta^-\{i\} = \{(ji) \in A \vert \forall j \in V\} \subset A$	is the set of "incoming" **arcs**
$\delta\{i\} = \delta^-\{i\} \cup \delta^+\{i\}$	is the set of **arcs** connected to i

We consider some notations on sets:

$\bar{S} = V \setminus S$ for $S \subset V$	is the complementary of a set S of vertices
$A(S) = \{(ij) \in A \vert \forall i,j \in S\}$ for $S \subset V$	is the set of arcs induced by a subset S of vertices
$\left(S : \bar{S}\right) = \{(ij) \in A \vert \forall i \in S, j \notin S\}$ for $S \subset V$	is the set of outgoing arcs of a set S
$x(E) = \sum_{(ij)\in E} x_{ij}$ for $E \subset A$ and $x \in \mathbb{R}^{\vert A\vert}$ a vector of reals	

Distances We consider throughout this report:

d_{ij} as the shortest distance between vertices i and j in the warehouse.

$d(E) = \sum_{(ij)\in E} d_{ij}$ for $E \subset A$

$d_x = \sum_{(ij)\in A} d_{ij} x_{ij}$ for $x \in \mathbb{R}^{\vert A\vert}$ a vector of reals

```
nbAisles = 5;
nbCrossAisles = 3;
nbOrders = 3;
nrPlacesPerRack = 6;
aisleWidth = 2;
rackDepth = 2;
locationWidth = 1;
crossAisleWidth = 3;

warehouseX = [5 6 6 ];

warehouseY = [2 5 1];

locationsX = [10.0 10.0 14.0 14.0];

locationsY = [19.5 5.5 8.5 4.5];

distances = [
[0 14.0 15.0 19.0]
[14.0 0 11.0 11.0]
[15.0 11.0 0 4.0]
[19.0 11.0 4.0 0 ]
];
```

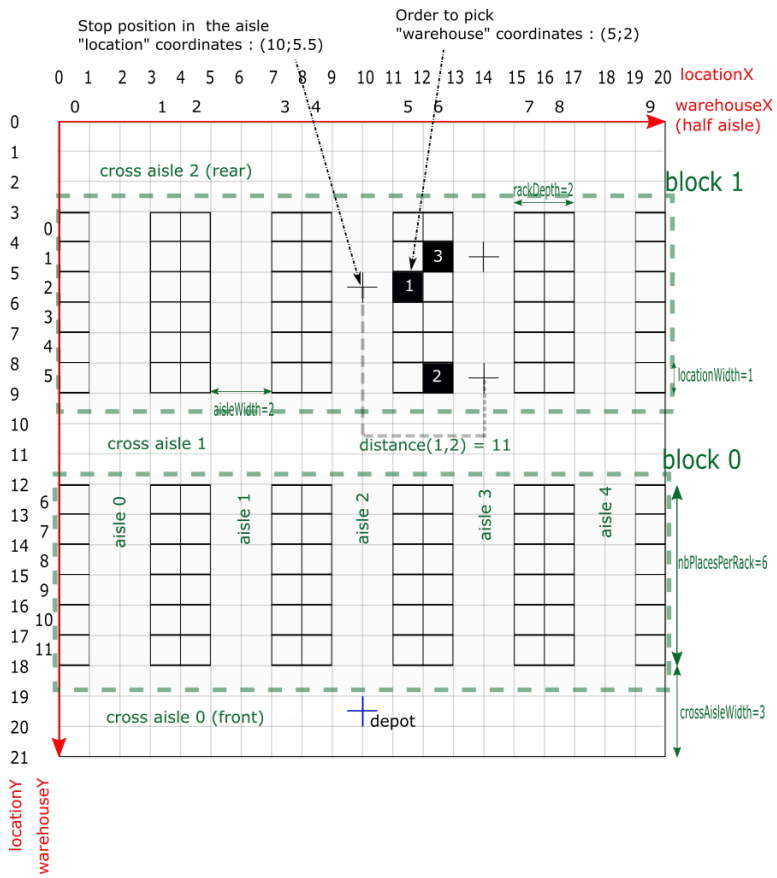


Figure 1: An example of data description and its representation

Warehouse features The following definitions are important all along this report:
$\mathbf{R} = \{0, 1, ...n\}$ is the set of locations to visit: 0 is the depot, $1, ..., n$ are the products.
$\mathbf{I}$ is the set of all intersections in the warehouse.
$\mathbf{V} = I \cup R$

$P_{ij} := \{P : (ij)-$shortest path in the warehouse $P = (iv_1v_2...j), v_i \in V\}$, $\forall i, j \in R$ denotes the set of all shortest paths between two locations in the warehouse.
For sake of simplicity, if (uv) is an arc contained in at least one shortest path between i and j, we write $(uv) \in P_{ij}$ (instead of $\exists P \in P_{ij}/(uv) \in P$).

A warehouse is defined mainly by its number of cross-aisles h and its number of vertical aisles v. The **size** of a warehouse is given by $h \times v$.

3 State of the art

For any configuration of a warehouse, the order picking problem is a special case of the Traveling Salesman Problem, $\mathcal{NP}$-hard [25]. This problem, introduced by Dantzig, Fulkerson and Johnson in 1954 [12], is one of the most studied in Operations Research. The survey of Orman and Williams [33] gives an overview of integer programming formulation for the TSP. The TSP, even if it's a $\mathcal{NP}$-hard problem can be solved quite quickly by Concorde which is one of the best exact solvers for the TSP (according to Hahsler and Hornik [21] or Mulder and Wunsch [30] for example).

To solve the specific case of picking problem, many heuristics have been proposed in particular by Hall [22]. Some performance analysis of the most popular heuristics were made by Petersen [34] and by Roodbergen and De Koster [37]. Theys, Bräysy, Dullaert and Raa [39] proposed to combine classical TSP heuristic with picking heuristic and provided a benchmark which is used in this work.

A specific exact approach by dynamic programming has been proposed for the first time by Ratliff and Rosenthal in the case of a single block in 1983 [36]. This algorithm was extended by Roodbergen and De Koster [38] in the case of three cross-aisles. These algorithms are polynomial in the number of aisles and products of the warehouse.
Catusse and Cambazard extended it to the general case of any warehouse, but this algorithm is exponential in the number of cross-aisles. Actually, this extension holds for any rectilinear TSP[7].

De Koster and Van der Poort [14] compared the initial dynamic programming with the commonly used S-shape heuristic and showed that "the numerical results suggest that the savings in travel time may be substantial when using the optimal algorithm instead of the S-shape heuristic". This result strengthens the will of finding exact and efficient algorithms to solve this problem.

In this study, we propose exact algorithms based on Mixed Integer Linear Programming (MILP) and modeling the problem either as a TSP or a Steiner TSP. The Steiner TSP is a variant of the TSP which was proposed independently by Fleischmann [15] and Cornuéjols, Fonlupt and Naddef in 1985 [11] even if Orloff introduced the idea some years before [32]. Burkard *et al.* [6] categorized this problem in well-solvable special cases of the TSP, especially in the case of series-parallel graph [11]. We used the compact MILP formulations proposed by Letchord, Nasiri and Theis [27].

4 Industrial application

The order picking problem is not solved optimally in current enterprise resource plannings (ERP for short) or warehouse management systems (WMS) because the representation of data and the resolution are complex. However, it is a big lever to reduce logistical costs so it seems interesting to find some algorithms and methods to enable firms to reduce their warehouse costs. It is a key step in the supply chain since it accounts for 55% of the total operational warehouse costs (see Figure 2) .

One track to improve software used in logistics is to include a linear solver to enable optimization of many problems. Actually, the solver ILOG is already available in the famous

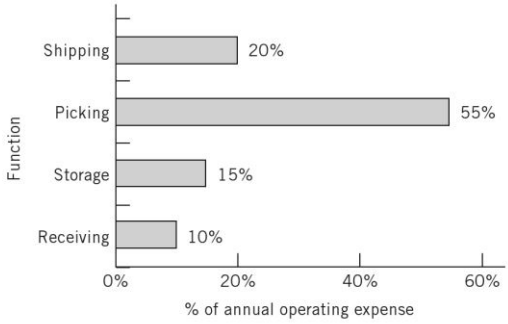


Figure 2: Typical distribution of warehouse operating expenses [40]

software SAP. The integration of linear solver allows the implementation of generic and flexible approaches: it is easy to add side constraints and to construct models that fit the needs of the users.

Many strategies are possible to improve the efficiency of the order picking process, at tactical or operational level:

- Layout design aims to efficiently organize the warehouse (tactical level).
- Storage policies are meant to efficiently store products in the warehouse. Typically, it seems interesting to have a volume policy, this means to gather products that are often picked together (tactical and operational level).
- Order consolidation policies are concerned with getting a picking list from customer orders. An optimized strategy is to batch orders when they are small enough and "close" enough (tactical and operational level).
- Routing policies search for the best way to move inside the warehouse, so that the order picker collects all the products as quickly as possible (operational level).

All these strategies are closely linked and can have a huge impact on travel time in the warehouse [35]. In this work, we focus on routing strategies which consists in minimizing the travel time spent in the warehouse.
Note that the time spent by an order picker in a warehouse is not only dedicated to the travel (see Figure 3). Indeed, the order picker may search for a product in the slots, then he has to pick the products and place them in a basket or a pallet. He has also administrative tasks such as validating orders. However, the travel part is the most important of these different costs [13, 40].

In practice, the warehouse routing problem is mainly solved by heuristics. The most commonly used in the case of multiple cross-aisles, are presented in Figure 4:

- S-Shape (or transversal): any sub-aisle containing a product is traversed through the entire length.
- Largest gap: the order picker enters a sub-aisle as far as the largest gap (largest distance between two items or an item and an intersection).
- Aisle-by-aisle: all the products of an aisle are picked in one go, then the order picker changes aisle by taking the best (calculated by dynamic programming) cross-aisle.
- Combined: beginning at the block at the rear of the warehouse, the order picker visits each sub-aisle sequentially, then changes block.

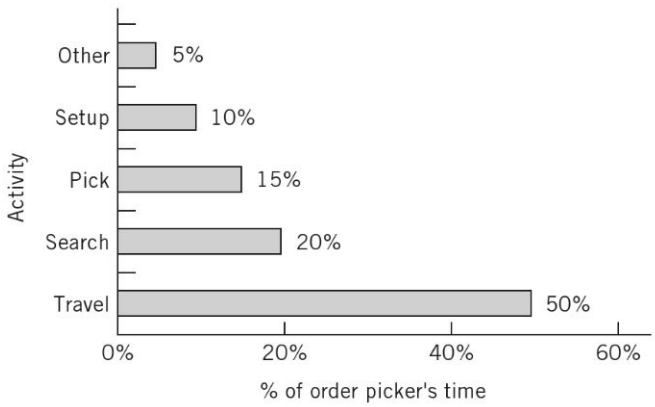
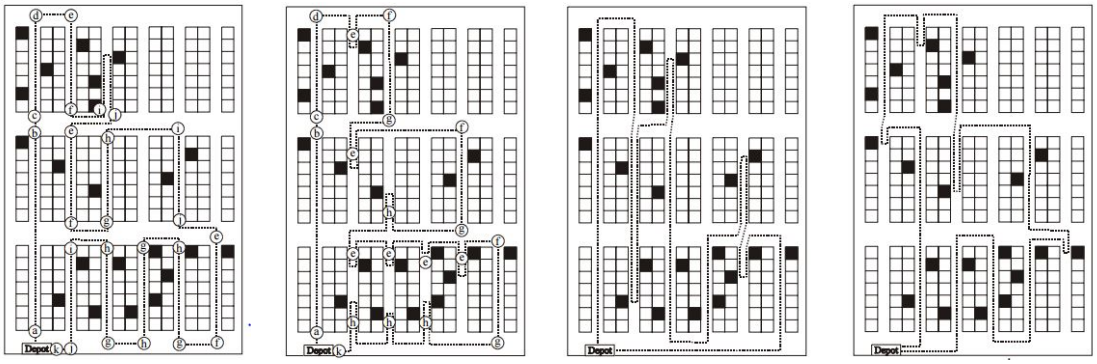


Figure 3: Typical distribution of an order picker's time [40]



(a) S shape (b) Largest gap (c) Aisle by aisle (d) Aisle by aisle

Figure 4: Example of routing methods [37]

The quality of these heuristics is not guaranteed and may be far from the optimum[14]. Thus, we propose exact approaches to find the optimal travel route in the warehouse.

5 Standard TSP formulation

An idea to compute the optimal tour in the warehouse is to solve a standard TSP where the distance between two vertices is given by the shortest path in the warehouse. Namely, we consider a complete graph, where the vertices represent orders (including the depot). Every vertex has to be visited exactly once, by minimizing the travel distance. The distance between two vertices is given by d_{ij}.

5.1 Conventional formulation

The conventional formulation, or sub-tour elimination formulation, due to Dantzig, Fulkerson, and Johnson [12] of the TSP is the following :

We define a variable x'_{ij} for each pair of products: $\forall i, j \in R$

$$x'_{ij} = \begin{cases} 1 \text{ if the tour uses the arc } (ij) \\ 0 \text{ otherwise} \end{cases}$$

$$
(DFJ) \qquad
\begin{aligned}
min \quad & \sum_{i,j \in R} d_{ij} x'_{ij} && && (1)\\
s.t. & \sum_{j \in R} x'_{ij} = 1 && \forall i \in R && (2)\\
& \sum_{j \in R} x'_{ji} = 1 && \forall i \in R && (3)\\
& x'\left(S : \bar{S}\right) \geq 1 && \forall S \subset R : 2 \leq |S| \leq \frac{|R|}{2} && (4)\\
& x'_{ij} \in \mathbb{N} && \forall i, j \in R && (5)\\
& && && (6)
\end{aligned}
$$

Constraints (2) and (3) impose that the order picker comes exactly once at each product and leaves them exactly once.
Constraints (4) impose that for any partition into two subsets S and $\bar{S}$, the order picker transits from S to its complementary at least once.
Due to the exponential number of sets to consider in constraints (4), the implementation of such a formulation follows a **cutting plane algorithm**. This method, proposed by Gomory [18], consists in solving a relaxation of a linear program. If the solution found does not respect the constraint that has been relaxed, an inequality is violated. This inequality is a *cut* and is added to the program. This process is repeated until a feasible solution is found.

5.2 A *branch-and-cut* approach: Concorde

One of the best, if not the best, solvers for the TSP is Concorde [1] (freely available for academic use). Even if the use of Concorde in industry is often not conceivable (due to the impossibility of adding side constraints and the cost of the software), we study the quality of the resolution with Concorde to compare our algorithms with a "state of the art" of the TSP resolution.

This solver has a complex algorithm based on LP relaxation, *branch-and-cut*, valid inequalities and heuristics.
In fact, the algorithm follows a *branch-and-bound* scheme, embedding cutting planes in it: this is the *branch-and-cut* approach. At each node of the tree, the algorithm adds some inequalities.

Many of these inequalities have been studied and classified: sub-tour cuts [12], comb inequalities [19, 9], star cuts [16, 31], clique-tree and bipartition [5, 20] and domino-parity [26]. To get further details, refer to Applegate *et al.* bibliography ([2, 3] in particular).
We will call this approach CDE.

5.3 Flow-based formulation

To avoid the exponential number of constraints (4) that requires complex cutting planes techniques, so called "compact formulations" have been designed. We investigate here the one of Gavish and Graves [17]. This formulation circulates a flow in the graph: the order picker leaves the depot with n units of a commodity and delivers one unit each time he picks an item.

We add, to the previous x'_{ij}, the variables y'_{ij} giving the amount of commodity passing through arc (ij) $\forall i, j \in R$.

The solution of the TSP is then found by solving the following mixed-integer linear program:

$$
(SCF)\quad
\begin{array}{llll}
z'^* = min & \sum_{i,j\in R} d_{ij}x'_{ij} & & (7) \\
\text{s.t.} & \sum_{j\in R} x'_{ij} = 1 & \forall i \in R & (8) \\
 & \sum_{j\in R} x'_{ji} = 1 & \forall i \in R & (9) \\
 & \sum_{j\in R} y'_{ji} - \sum_{j\in R} y'_{ij} = 1 & \forall i \in R \setminus \{0\} & (10) \\
 & y'_{ij} \leq n x'_{ij} & \forall i,j \in R & (11) \\
 & x'_{ij} \in \{0;1\} & \forall i,j \in R & (12) \\
 & y'_{ij} \geq 0 & \forall i,j \in R & (13)
\end{array}
$$

Assignment constraints (8), (9)

Flow delivery (10)

Bound on commodity amount (11)

Constraints (8) and (9) are usual assignment constraints ensuring that each vertex is visited **exactly** once.
Constraints (10) ensure that, except for the depot, the salesman deliver one unit at each vertex and retains the rest of the flow.
Constraints (11) are *Big-M* constraints linking y and x so that if some flow transits through (ij) then the arc (ij) is chosen. Finally, the objective (7) is to minimize the total cost of the tour.

Remark. The variables y' are real variables but the optimal solution will be integer due to the fact that y' represent a flow.

The drawback of these formulations is that we completely loose the structure of the warehouse and consider a complete graph while the initial one is really sparse. As a result, this model is unable to scale to solve realistic instances. Therefore, in the following, we extract some properties from the problem structure and investigate alternative formulations.

6 Steiner formulation

The Steiner variant of the TSP was proposed by Cornuéjols, Fonlupt and Naddef in 1985 [11]. This variant was introduced especially to solve problem where the graph is sparse. The principle is that the graph contains some **required vertices** which **must** be visited and some **Steiner vertices** which **can** be visited. Moreover, in a Steiner Traveling Salesman Problem (STSP), the graph is not complete and edges as well as **vertices can be visited more than once**.
In this section, we apply a Steiner approach to the picking problem.

6.1 Construction of the Steiner graph

To transform an instance of the picking problem to an instance of the Steiner STSP we need to define the directed graph $\mathbf{D} = (\mathbf{V}, \mathbf{A})$ (see Figure 5):

R is the set of required vertices.
I is the set of Steiner vertices.
Let $\mathbf{V} = \mathbf{R} \cup \mathbf{I}$ be the set of all vertices.

Let $\mathbf{A}$ be the set of possible arcs: $\forall i, j \in V,\ (i,j) \in A \Leftrightarrow$ one of the following conditions holds:

i. i and j are horizontally adjacent intersections (e.g., arc 4-5 in Figure 5).

ii. i and j are extreme intersections of an empty sub-aisle (e.g., arc 4-9 in Figure 5).

iii. j is an extreme products and i the adjacent intersection (e.g., arc 3-16 in Figure 5).

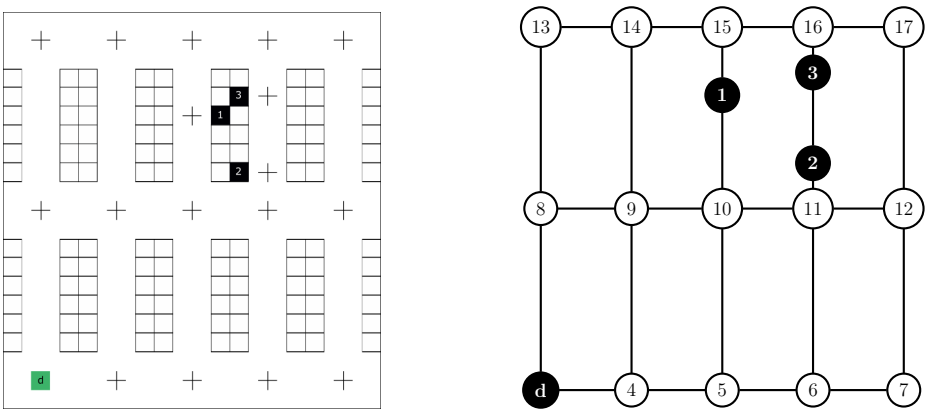


Figure 5: Graph representation of the Steiner TSP induced by the instance described in Figure 1 . The black vertices are the required ones. d is the depot.

iv. i and j are adjacent products (e.g., arc 2-3 in Figure 5).

Definition 1. The graph D is such that $\forall (ij) \in A : (ji) \in A$ and $d_{ij} = d_{ji}$. We say that D is *symmetric*.

Remark. Since D is symmetric, $\Gamma\{i\} := \Gamma^+\{i\} = \Gamma^-\{i\}\ \forall i \in V$

Definition 2 (Facing products)**.** Two products are **facing** each other if they are in the same aisle, at the same ordinate but one lies in the left slot whereas the other one lies in the right slot.

From hypothesis 1, the distance between facing products is null. Thus, they are represented by coincident vertices in the Steiner graph.

Remark. By construction, each vertex as **at most 4 neighbors**.

6.2 Flow-based formulation

We can use a compact Steiner formulation in mixed-integer linear programming proposed by Letchford, Nasiri and Theis [27]. It follows the flow principle: the order picker leaves the depot with n units of a commodity and delivers one unit each time he picks an item.

We define the variables: $\forall (ij) \in A$

$$x_{ij} = \begin{cases} 1 \text{ if the tour uses the arc } (ij) \\ 0 \text{ otherwise} \end{cases}$$

y_{ij} = amount of commodity passing through arc (ij).

The solution of the STSP is then found by solving the following mixed-integer linear program:

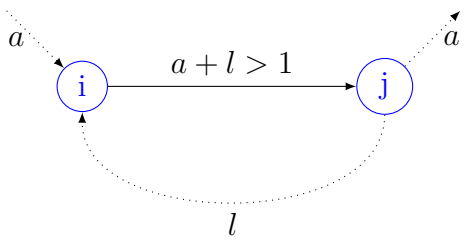
$$
(SCFS)\quad
\begin{array}{llllll}
z^* = min & \sum\limits_{(ij)\in A} d_{ij}x_{ij} & & (14) & \\
\text{s.t.} & \sum\limits_{j\in\Gamma\{i\}} x_{ij} \geq 1 & \forall i \in R & (15) & \text{Assignment constraints} \\
& \sum\limits_{j\in\Gamma\{i\}} x_{ij} = \sum\limits_{j\in\Gamma\{i\}} x_{ji} & \forall i \in V & (16) & \text{Conservation} \\
& \sum\limits_{j\in\Gamma\{i\}} y_{ji} - \sum\limits_{j\in\Gamma\{i\}} y_{ij} = 1 & \forall i \in R \setminus \{0\} & (17) & \text{Flow delivery} \\
& \sum\limits_{j\in\Gamma\{i\}} y_{ji} - \sum\limits_{j\in\Gamma\{i\}} y_{ij} = 0 & \forall i \in V \setminus \{R\} & (18) & \text{Flow conservation} \\
& y_{ij} \leq n x_{ij} & \forall (ij) \in A & (19) & \text{Bound on commodity amount} \\
& x_{ij} \in \mathbb{N} & \forall (ij) \in A & (20) & \\
& y_{ij} \geq 0 & \forall (ij) \in A & (21) &
\end{array}
$$

Constraints (15) ensure that each required vertex is visited **at least** once. Constraints (16) ensure that the tour arrives in any vertex as many times as it leaves it.
The flow constraints are different depending on whether a vertex is required or not: constraints (17) impose that the order picker delivers one unit of the commodity to each product while constraints (18) impose that the flow stays the same through a non-required vertex.
Constraints (19), as previously, link the flow y to the x variables.

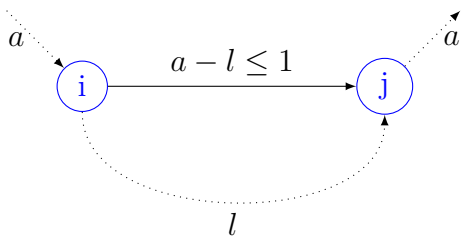
Remark. As before we relaxed integrity constraint on variables y.

Remark. We know from Lemma 1 in Letchford, Nasiri and Theis [27] that every optimal solution of the STSP will respect $x_{ij} \leq 1\ \forall (ij) \in A$. It is sufficient to define x as a positive integer, as this result is also true when we consider the relaxed solution.

Proof. Consider a fractional optimal solution obtained by solving the linear relaxation of (SCFS). Assume that there exists an arc (ij) such as $x_{ij} > 1$. This means we have the following scheme (dotted arcs represent paths) :



Without loss of generality, we can consider the smallest subgraph having this structure and then assume that $a \leq 1$, $l \leq 1$ and $x_{ij} = a + l > 1$. Thus, we can change the solution so that $x_{ij} \leq 1$ by changing the direction of the path between j and i :



The new solution is feasible and costs strictly less than the first solution, which was therefore not optimal. We have a contradiction and so we proved that $x_{ij} \leq 1 \quad \forall (ij) \in A$ ■

6.3 Strengthening of the bound

Constraints (19) are big-M constraints and thus, make the formulation quite weak. Indeed, the linear relaxation of MILP models with big-M constraints are known to be very poor (see Codato and Fischetti for example [10]). However, the smaller is the value of "M" the stronger is the linear relaxation.

Without loss of generality, we can assume that the order picker delivers the unit of commodity due to each required vertex the first time it enters it. Then we can deduce from the warehouse structure a minimum number of required predecessors for each vertex, let's call it $n_R(i)$. To reach a vertex i, we can know in advance that at least $n_R(i)$ required vertices have to be traversed, and thus we can reinforce the bound on y_{ij} ($\forall j \in \Gamma^+\{i\}$).
We apply a Bellman-Ford algorithm and compute, for every vertex i, the minimum number of required vertices (except depot since no unit is delivered in it) that have to be visited before the order picker leaves i (including himself).
We can then replace (19) by:

$$y_{ij} \leq (n - n_R(i))\, x_{ij} \;\forall i \in V, j \in \Gamma\{i\} \tag{22}$$

6.4 Other formulations

We chose the SCFS formulation after a preliminary study of other formulations proposed by Letchford, Nasiri and Theis [27] which are: Fleischmann or DJFS (as it is an adaptation of the classical formulation, this means with the sub-tours elimination [15]), Multi-Commodity Flow formulation (MCFS) and Time-Staged formulation (TSS).

Table 1 shows the size of each formulation.

	SCFS	DFJS	MCFS	TSS
Variables	$\mathcal{O}(\|A\|)$	$\|A\|$	$\mathcal{O}(\|R\|\|A\|)$	$\mathcal{O}(\|V\|\|A\|)$
Constraints	$\mathcal{O}(\|A\|)$	$\mathcal{O}(2^{\|R\|})$	$\mathcal{O}(\|R\|\|A\|)$	$\mathcal{O}(\|V\|^2)$

Table 1: Alternative STSP formulations and their size (see [27]). We recall that R are the required vertices, V the set of all vertices and A the set of arcs in the Steiner graph.

The conjectures on the lower bounds given by the linear relaxation of each formulation are [27]:

$$\text{SCFS}_{LR} \leq \text{TSS}_{LR} \leq \text{DFJS}_{LR} = \text{MCF}_{LR}$$

So the single flow formulation is dominated by the time-staged formulation, which in turn is weaker than the classical sub-tours elimination models.

Despite the strength of these other formulations, we did not work on them because they were not compact enough to scale up, even after the preprocessing proposed in the next section. However, the extended formulation will be used to get some theoretical results. Fleischmann formulates the problem as the following integer linear program:

We define the variables: $\forall (ij) \in A$:
$x_{ij} = \begin{cases} 1 \text{ if the tour uses the arc } (ij) \\ 0 \text{ otherwise} \end{cases}$

$$
(DFJS) \qquad
\begin{aligned}
min \quad & \sum_{(ij)\in A} d_{ij}x_{ij} && && (23)\\
s.t. \quad & \sum_{j\in\Gamma\{i\}} x_{ij} = \sum_{j\in\Gamma\{i\}} x_{ji} && \forall i \in V && (24)\\
& x\left(S : \bar{S}\right) \geq 1 && \forall S \subset V : S \cap R \neq \emptyset, R \setminus S \neq \emptyset && (25)\\
& x_{ij} \in \mathbb{N} && \forall (ij) \in A && (26)
\end{aligned}
$$

As before, this sub-tour elimination formulation must be implemented by a cutting plane algorithm. However, in this case, due to the big number of redundant cuts, the convergence is really slow. Indeed, there exists a lot of different sets $S \subset V$ which partition the required vertices in the same way but partition differently the Steiner vertices.

The following section presents our work to keep the small size of the SCFS formulation and to improve the quality of its linear relaxation.

7 Improvement of formulations

A MILP is solved by a branch-and-bound algorithm. The resolution time of a MILP is impacted by two things: the size of the search space and the quality of the linear relaxation which is a lower bound on the optimal value. So we can reduce resolution time by reducing the space of search, this means by creating fewer variables and by improving the linear relaxation to have the smallest gap possible between this lower bound and the optimal value. Thus, we can improve the preceding formulations by exploiting the problem structure.

We first propose some way to preprocess the graph to reduce the number of variables in the MILP. Then we present some cuts to improve the linear relaxation and thus the resolution of the problem.

7.1 Preprocessing

Each formulation solves a MILP where variables are related to the layout of the warehouse. Some work can be done on this layout to have a smaller input and thus, have fewer variables. We show different ways to suppress vertices and arcs without loosing optimality.

7.1.1 Vertex preprocessing

In this section, we reduce the number of vertices without changing the problem thanks to its specific structure.

First, in each formulation, it's immediate from hypothesis 1 that when two products are facing each other, we can keep only one (see def. 2)

Second, Lemma 1 from Letchford, Nasiri and Theis [27] saying that an arc is taken at most once implies that the order picker enters a sub-aisle at most twice, and cannot make more than one U-turn by entrance. Thus, we can know in advance some sets of products that have to be picked together in an optimal solution. We recall that a sub-aisle contains all items of one aisle in one block and the two intersections surrounding them.

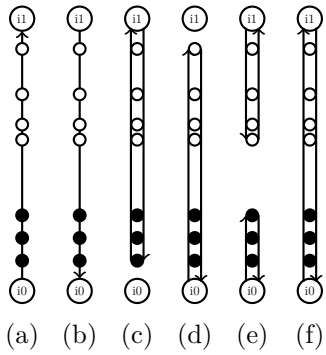


Figure 6: The six ways to traverse a sub-aisle. We distinguish two groups (black and white vertices) where all vertices of one group are taken in one go.

Definition 3 (Largest gap)**.** The largest gap of a sub-aisle is the longest empty distance between two vertices of a sub-aisle.

The largest gap can be between a product and an intersection and it may not be unique.

Thus, there exists only six unique ways to traverse a sub-aisle, shown in Figure 6.

It's an extension of the results of Ratliff and Rosenthal[36] in the case where arcs are directed and there are more than 2 cross-aisles (which allows 6(f)) .

Case 6(e) is the only configuration that may not be unique but only the best ones (i.e., the ones where the edge which is not taken is a largest gap) will occur in an optimal solution and no matter which one of the largest gap is taken the solution will have the same value.

Cases 6(a), 6(b) and 6(f) are allowed even if no products has to be picked in the sub-aisle. Indeed, the order picker may have to traverse empty sub-aisle to reach products located in higher blocks.

In any case, the black vertices are all taken together and the white vertices as well. So, for both subsets we can keep only extreme products and impose an arc to be taken between these two products.

Definition 4 (Preprocessing)**.** The vertex preprocessing is defined by the following algorithm and shown in Figure 7:

for *every sub-aisle* **do**

Compute a largest gap

Identify the set S (resp. T) containing all products below (resp. above) the largest gap

In each subset (S and T), keep the two products that are the farthest t_S and b_S (resp. t_T and b_T)

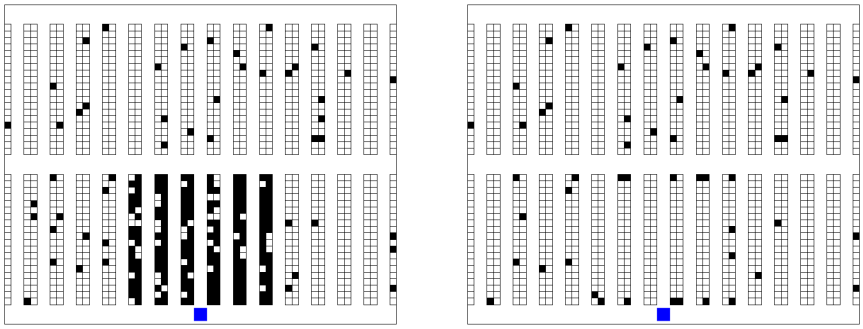
Add the constraints:
$x_{t_S b_S} + x_{b_S t_S} \geq 1$ if $t_S \neq b_S$
$x_{t_T b_T} + x_{b_T t_T} \geq 1$ if $t_T \neq b_T$

end

Remark. S and T can be empty or singletons (and then $t_S = b_S$ and resp.).

Figure 7 shows the result after preprocessing on sub-aisle of Figure 6. Set S are black vertices, set T white vertices.

Figure 7: Configuration of a preprocessed sub-aisle. In the model, dashed products are not taken and edges are forced at 1.



(a) Number of (product) vertices: 240 (b) Number of (product) vertices: 54

Figure 8: Before and after preprocessing

Note that the six configurations have either $(t_S b_S)$ or $(b_S t_S)$ (or both) in one side and either $(t_T b_T)$ or $(b_T t_T)$ (or both) in the other side, whence the constraints.

Remark. With this preprocessing, each sub-aisle contains at most four products.

Proposition 1. *The preprocessing does not change the value of the optimal solution and leave* ***at most 4 products*** *by sub-aisle.*

Figure 8(b) shows the result after the preprocessing step on a specific instance where products are stored with a volume policy. We clearly notice that the number of products is significantly reduced which implies a really smaller number of vertices than in Figure 8(a).

Adaptation of vertex preprocessing to Concorde input

This preprocessing cannot be applied when using Concorde because we cannot add constraints: Concorde takes as input only a matrix of distances.
However, the constraints which are added in the preprocessing are required only if the largest gap changes in the sub-aisle. Thus, we still can suppress a few vertices if we keep the largest gap at the same place, since a solution given by Concorde respects the six ways to traverse sub-aisle, as every optimal solution.
Figure 9 shows that if we keep the same products than the preceding preprocessing, the problem changes: as the largest gap stands where we suppressed vertices, these vertices may not be picked in the configuration 6(e). In the second case, we kept products such that the largest gap remains unchanged so the problem also remains unchanged.

To suppress the maximum number of vertices, we keep the vertices the farthest from the extreme products of the sets S and T and suppress the others as long as the largest gap stays

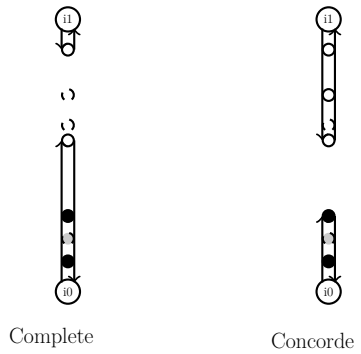


Figure 9: Pattern 6(e) depending on the preprocessing. The complete preprocessing leads to a wrong solution.

unchanged.

7.1.2 Arcs preprocessing

Once the graph has the minimum number of required vertices we can notice that for any pairs of vertices, there can exist several shortest paths. The purpose of this part is to keep edges that are know sufficient to contain **an** optimal solution. For this, we compute the minimum 1-spanner in terms of number of edges, i.e., we are looking for a subset of arcs preserving at least one original shortest path between any pairs of required vertices.
Indeed, any solution of the picking problem gives a tour where two items picked successively are linked by a shortest path. So in any 1-spanners, an optimal solution is possible.

Definition 5 (k-spanner)**.** A **k-spanner** of a graph G is a sub-graph $H \subset G$ such that:

- H contains all the vertices of G.
- The distance between each pair of vertices in H is at most k times their distance in G.

In our case, we search for a minimum "1-spanner" in the Steiner graph, restricted to the required vertices.
Let's consider the undirected graph $G = (V = I \cup R, E)$ where E is defined as A by removing orientation.
We search for a subgraph of G such that between each pair (ij) of required vertices there exists a shortest path which keeps the initial cost d_{ij}.
Namely we search a graph $H = (V_H, E_H)$ such that: $V_H \supset R$, $V \supset V_H$, $E_H \subset E$ and
$\forall i, j \in V_H : \exists$ a (i, j)-path $P \in H / d(P) = d_{ij}$.

To compute a minimum 1-spanner, we choose the minimum set of edges with respect to the preceding properties. To be more efficient, we can remove the edges which don't belong to any shortest path between required vertices and compute the minimum 1-spanner on the retained edges.

Definition 6 (Interval)**.** An interval between two vertices is the set of all edges used in any shortest path between these two vertices.

Appendix B shows some examples of intervals depending on the position of the two vertices.

We use an integer linear program to solve the problem of the **1-spanner**. Let E' be the set of edges of all the intervals between required vertices. Let B_{ij} be the set of edges of the interval between the vertices $i, j \in R$

We define the variables:

$$a_e = \begin{cases} 1 \text{ if the edge } e \text{ is kept in the 1-spanner} \\ 0 \text{ otherwise} \end{cases} \qquad \forall e \in E'$$

$$b_{ij,e} = \begin{cases} 1 \text{ if the edge } e \text{ is kept in the shortest path between } i \text{ and } j \\ 0 \text{ otherwise} \end{cases} \qquad \forall i, j \in R \; \forall e \in B_{ij}$$

$$\begin{array}{llll}
min & \sum_{e \in E'} a_e & & (27) \quad \text{minimization of the number of edges} \\
\text{s.t.} & \sum_{e \in \delta(i)} x_e \geq 1 & \forall i \in R & (28) \quad \text{reach each required vertex} \\
& \sum_{e \in B_{ij} \setminus (\delta(i) \cup \delta(j))} b_{ij,e} = 0 & \forall i, j \in R & (29) \quad \text{flow conservation} \\
& \sum_{e \in B_{ij} \cap \delta(i)} b_{ij,e} = 1 & \forall i, j \in R & (30) \quad \text{flow input} \\
& \sum_{e \in B_{ij} \cap \delta(j)} b_{ij,e} = -1 & \forall i, j \in R & (31) \quad \text{flow delivery} \\
& \sum_{e \in B_{ij}} d_e b_{ij,e} = d_{ij} & \forall i, j \in R & (32) \quad \text{distance conservation} \\
& b_{ij,e} \leq a_e & \forall i, j \in R, e \in B_{ij} & (33) \\
& a_e \in \{0; 1\} & \forall e \in E' & (34) \\
& b_{ij,e} \in \{0; 1\} & \forall (i, j) \in R, e \in B_{ij} & (35)
\end{array}$$

Constraints (28) impose that each required vertex must be reached by a selected edge.

Variables $b_{ij,\cdot}$ represent a single-commodity flow circulating from the source i toward the sink j. For each pair of required vertices ij, constraints (30) impose that vertex i introduces one unit of a commodity and constraints (31) impose that vertex j receives this unit. Constraints (29) then ensure that the flow is kept all along the path.
Constraints (32) certify that the selected edges between i and j respect the shortest distance.
Finally constraints (33) guarantee that a flow can circulate on an edge only if it is kept.

Actually, constraints (32) are not necessary since the flow has to be kept along a shortest path and, by the objective function, no more than the necessary number of edges will be taken.

Remark (Mandatory edges). When the shortest path is unique between two vertices, we can set $a_e = 1$ for each edge e of this path.

The resulting set of edges $E^* = \{e \in E' : a_e = 1\}$ forms the 1-spanner which can be used to compute the STSP.

Remark (Empty interval). In fact we don't have to maintain a shortest path between every pairs of required vertices but only when the interval generated by the pair is **empty**, i.e. does not contain any other product.
Indeed, imagine two products p_1 and p_2. If one of the shortest path between them crosses the product p_3 then it is sufficient to have a shortest path between p_1 and p_3 and between p_3 and p_2.

Definition 7 (Empty interval)**.** An interval is **empty** if no shortest path crosses a required vertex.
The **generating set** is the set of all empty intervals.

Thus, the variables are defined only for $i, j \in R$ such that B_{ij} is empty.

Remark (Complexity). This problem is in fact closer to a problem of Minimum Manhattan Network [4] (1-spanner in the rectilinear case) which is $\mathcal{NP}$-hard [8]. This may be an issue in our resolution but in practice the resolution is really quick.
Moreover, we don't need to compute the optimal network since it is just a preprocessing to reduce the number of edges. Actually, we could keep the generating set or apply a simple approximation.

Improvement brought by the preprocessing

These different preprocessings improve the quality of the formulation, in that the linear relaxation is better. Moreover, it reduces, sometimes drastically, the size of the problem. It becomes linear in the size of the warehouse.
After the vertex preprocessing we have $|R| \leq 4hv$.
After the computation of the 1-spanner, we observe two benefits. First, the size of A is reduced. Second, the Bellman-Ford algorithm gives stronger bound since the shortest paths are computed in a sparser graph.

7.2 Sub-tours cuts

Once we made our Steiner Single Flow formulation as compact as possible, we want to improve its quality by increasing the linear relaxation. As we know that the sub-tour elimination formulation has the best linear relaxation, we expect that cutting sub-tours will enforce our formulation.

In the single-flow formulation, the connexity between all the required vertices is guaranteed. However, because of the big-M constraints (22), the fractional value of x can be really small in arcs connecting the graph. This implies that we can find many sets which doesn't respect constraints of sub-tours elimination(25).
In this section, we define some relevant sets, easily calculable, that must be balanced.

We focus on sets defined as cuts $(S, \bar{S})$ partitioning the warehouse into two subsets S, $\bar{S} \subset V$ where $S \cap R \neq \emptyset$ and $\bar{S} \cap R \neq \emptyset$. In other words, each subset contains at least one required vertex. Thus, there must be at least one arc going from S to $\bar{S}$ and at least one arc going from $\bar{S}$ to S.
For these cuts, we impose:

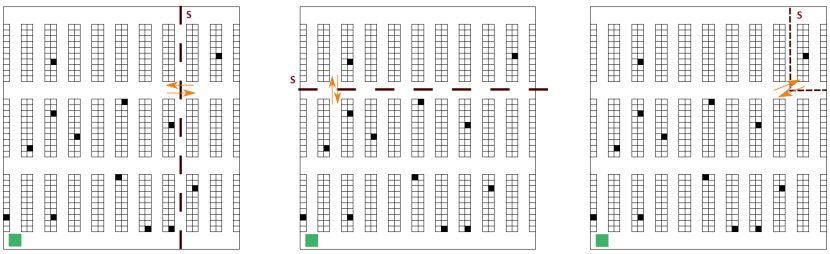
$$x\left(S : \bar{S}\right) \geq 1 \tag{36}$$

Remark. It is easy to see, with constraints of conservation (16) that this inequality is sufficient to impose also $x\left(\bar{S} : S\right) \geq 1$

7.2.1 Line cuts

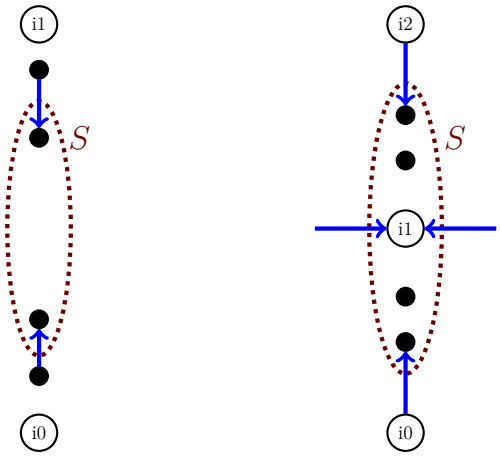
Let's define a horizontal (resp. vertical) cut $C = (S, \bar{S})$ separating the warehouse by cutting it on an aisle in the horizontal way (resp. on a sub-aisle in the vertical way) (see Figure 10).

These cuts are strong because they force the order picker to go through all the warehouse.



(a) Vertical cuts (b) Horizontal cuts (c) Corner boxes cuts

Figure 10: Line and corner boxes cuts



(a) Sub-aisle connexity (b) Cross cuts

Figure 11: Sum of arcs entering S must be greater than 1.

7.2.2 Corner boxes cuts

We can combine horizontal and vertical cuts to create boxes attached to a corner of a warehouse (see Figure 10(c)).

7.2.3 Sub-aisle connexity

In Figure 13(a), we can observe sub-tours between two products in a sub-aisle, as marked by ⓐ on the figure. To avoid these sub-tours, we impose on the order picker to enter the set of selected vertices.

We define $S \subset V$ as a set of adjacent vertices in the same sub-aisle (see Figure 11(a)). There exists at most 6 sets of adjacent vertices in a same sub-aisle since there are at most four products.

7.2.4 Cross cuts

Figure 12 presents the fractional value of x solution of the linear relaxation of $SCFS$. We can observe that $x(\bar{S}, S) < 1$ and $x(S, \bar{S}) < 1$.
We can avoid this kind of sub-tours by making "cross-cuts". In this case, it's sufficient to consider extreme products as shown in Figure 11(b).

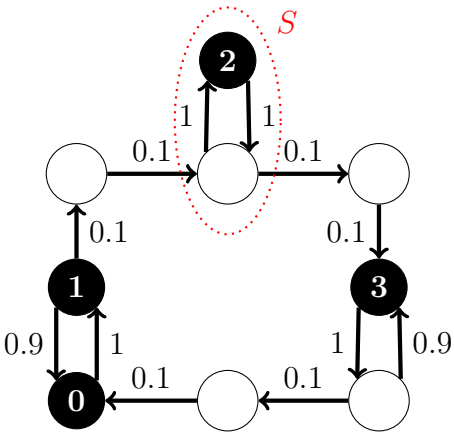
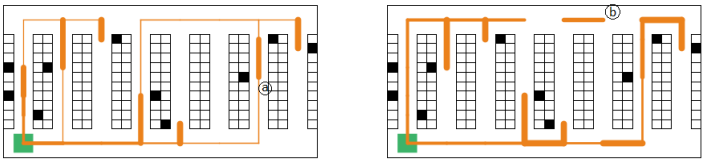


Figure 12: Example of a linear relaxation with a "cross" sub-tour S. Required vertices are black, Steiner vertices are white.



(a) Linear relaxation without the warehouse cuts = 79.5

(b) Linear relaxation with the warehouse cuts = 116

Figure 13: Example of impact of the sub-tour cuts (optimal value: 116). The thicker is the line, the bigger is the value of x on the arc

Figure 13 shows the impact of the sub-tours cuts on an example. The lines represent the value of the linear relaxation: the thicker is the line, the bigger is the value of x on the arc. We can see that adding the cuts can improve a lot the relaxation but induces the creation of a sub-optimal patterns ⓑ.
The total number of cuts added is polynomial in the size of the warehouse (complexity in $\mathcal{O}(hv)$). More precisely, we add at most:

Cuts	Number
Horizontal	h
Vertical	v
Corner boxes	$4(h-2)(v-1)$
Sub-aisles	$6(h-1)v$
Cross	$(h-2)v$

7.3 Dominance

All this section adds constraints on Steiner formulations that exploit the structure of the warehouse.

7.3.1 Intersection connexity

We want to prevent the relaxation to make sub-tours between two intersections as ⓑ in Figure 13(b).

Actually, for any intersection (except the depot), if an arc goes in it, it must continue the path to reach a product. Namely, if an arc comes in an intersection by one side, an arc must

go out by **another** side, this means to reach another vertex than the one it comes from. Thus, we add the following constraints:

$$x_{ij} \leq \sum_{k \in \Gamma(i) \setminus \{j\}} x_{ki} \quad \forall i \in I, j \in \Gamma\{i\}$$
$$x_{ji} \leq \sum_{k \in \Gamma(i) \setminus \{j\}} x_{ik} \quad \forall i \in I, j \in \Gamma\{i\}$$

7.3.2 Patterns

We know that there's only six ways to enter each sub-aisle (see Figure 6). From these patterns, we can identify some logical implications.

Let's study them on the general case with four products a, b, c, d in a sub-aisle surrounded by the intersections s and t. The largest gap is between b and c:

$x_{sd} \Rightarrow x_{dc}$ and $x_{ds} \Rightarrow x_{cd}$

$x_{ta} \Rightarrow x_{ab}$ and $x_{at} \Rightarrow x_{dc}$

$x_{cb} \Rightarrow x_{dc} \wedge x_{ba}$ and $x_{bc} \Rightarrow x_{ab} \wedge x_{cd}$

These logical implications are deduced from Figure 6.

For example, we observe that arc (sd) is taken only in cases (a), (c), (e) and (f). In all these cases, arc (dc) is also taken. Thus $x_{sd} \Rightarrow x_{dc}$

7.4 Cuts of symmetries

In a solution of the Steiner TSP, a vertex can be visited several times. Thus, there might be several ways to enter a vertex without changing the cost of the solution. For example, in Figure 14, going up or bottom first is strictly equivalent. This means that our formulations contains a lot of symmetries [29].

The symmetry is an important drawback in integer programming since it creates many isomorphic solutions in the search tree of the *branch-and-cut* and wastes time of search.

To avoid some of the symmetries, we can impose an order. Arbitrarily, we decide that, when the order picker is at an intersection and if he has the choice, he goes left, up, right and finally, down.

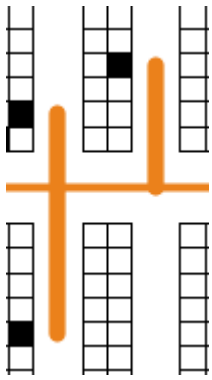
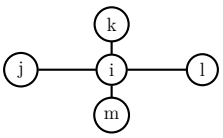


Figure 14

In the following example, the order picker is in i and we want to force him to go left first (only if he needs to go left of course).



The amount of flow circulating to a vertex is related to the number of products visited before. Thus, $y_{ij} \geq y_{il}$ means that (ij) is traversed before (il)). In our case, we want to impose that, if the arc (ji) is taken, it is taken before the other ones. Namely, the flow y traversing this arc has to be greater than the flow on the other edges (that might be null). This means:

$x_{ij} > 0 \Rightarrow y_{ij} \geq y_{ik}$	linearized into	$y_{ij} \geq y_{ik} + n(x_{ij} - 1)$
$x_{ij} > 0 \Rightarrow y_{ij} \geq y_{il}$	linearized into	$y_{ij} \geq y_{il} + n(x_{ij} - 1)$
$x_{ij} > 0 \Rightarrow y_{ij} \geq y_{im}$	linearized into	$y_{ij} \geq y_{im} + n(x_{ij} - 1)$

Thus, if $x_{ij} = 1$, we have the constraints $y_{ij} \geq y_{ik}$ and if $x_{ij} = 0$, y_{ij} is bounded from below by a negative number which is a valid inequalities in the model.
The same reasoning is applied to the rest of the order chosen.

Actually in practice, these new constraints do not have a clear positive impact on the resolution so we do not add them in our experiments.

8 Theoretical study of formulations

In this section we compare the quality of the different formulations described above. We define the polytopes of these linear programs, which represent the relaxed feasible solutions of our formulations. Thus, in all this section (expect preliminary remarks), we consider **real** variables.

Polytopes of flow-based formulations:

$$P_{SCF} = \left\{(x', y') \in \mathbb{R}^{|R|^2} \times \mathbb{R}^{|R|^2} \mid \ (x', y') \text{ satisfies (8) to (13) (page 9)}\right\}$$
$$P_{SCFS} = \left\{(x, y) \in \mathbb{R}^{|A|} \times \mathbb{R}^{|A|} \mid (x, y) \text{ satisfies (15) to (21) (page 11)}\right\}$$

Polytopes of sub-tour elimination formulations:

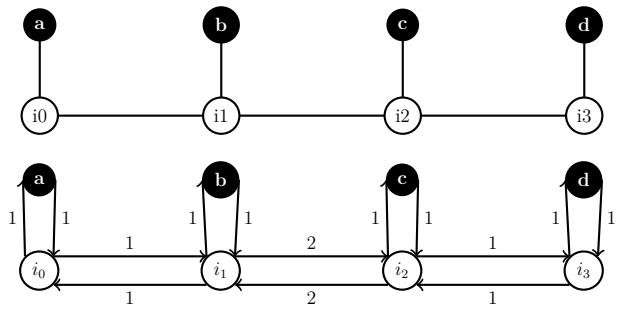
$$P_{DFJ} = \left\{x' \in \mathbb{R}^{|R|^2} \mid x' \text{ satisfies (2) to (5) (page 8)}\right\}$$
$$P_{DFJS} = \left\{x \in \mathbb{R}^{|A|} \mid x \text{ satisfies (24) to (26) (page 13)}\right\}$$

For all these sets, adding a * means we consider the sets of optimal solutions, the optimality being defined commonly by the minimization of the total distance covered.

8.1 Preliminary remarks

Remark (Equivalence of models). The sets of optimal solutions of the TSP and of the STSP are equal.

Proof. A solution is a permutation of the products. However, in a feasible solution of the STSP, the order picker can pass through a vertex several times while it is forbidden in the TSP.
Actually, if a vertex is visited a second time, it is on a shortest path between two other vertices and we can "remove" him from the permutation. If not, it means the tour makes an useless U-turn.



■

8.2 Projections

We want to compare our different models of the picking problem. We define here two mappings P_1 and P_2 between fractional solutions of the TSP and STSP models. These mappings are independent of the formulation and will be used to compare the quality (=the lower bound given by the linear relaxation) of the different formulations in the different models.

Firstly, we consider a fractional solution of the TSP (a solution of P_{SCF} or P_{DFJ}) and we give a mapping to a fractional solution of the STSP (a solution of P_{SCFS} or P_{DFJS}).

Definition 8 (Projection P_1 : from TSP to STSP)**.**

$$\begin{aligned} P_1 : \quad \mathbb{R}^{|R|^2} &\rightarrow \mathbb{R}^{|A|} \\ x' &\rightarrow x \end{aligned} \tag{37}$$

Where $P_1(x')$ is defined by:

$$x_{uv} = \sum_{i,j \in R} \sum_{P \in P_{ij} | (uv) \in P} \frac{x'_{ij}}{|P_{ij}|} \ \forall (uv) \in A \tag{38}$$

P_1 projects a solution of the standard TSP in the space of Steiner TSP by keeping the distance value. The idea is to divide the value x' between two required vertices on all the shortest paths between them is the Steiner graph.
By construction, the conservation is checked at each vertex (property (40)) and each required vertex is "visited" at least as many times as in the standard TSP solution (property (39)).

Remark (Properties of P_1). Let $x' \in \mathbb{R}^{|R|^2}$ and $x = P_1(x')$. Then:

(i)

$$\sum_{j \in \Gamma\{i\}} x_{ij} \geq \sum_{j \in R} x'_{ij} \ \forall i \in R \tag{39}$$

(ii)

$$\sum_{j \in \Gamma\{i\}} x_{ij} = \sum_{j \in \Gamma\{i\}} x_{ji} \ \forall i \in V \tag{40}$$

(iii)

$$d_x = d_{x'}$$

Secondly, we give the reversed mapping from STSP to TSP space. For this, we first define the following notions:

Definition 9 (Empty path between i and j)**.**
$P = (iv_1v_2...v_kj)$, with each $v_i \in V$ is an *empty path* between i and $j \in S$ with respect to the couple (x, S) if:

- $x(uv) > 0 \ \forall (uv) \in P$ (there is a positive quantity traversing the path)
- $v_l \notin S \ \forall l = 1...k$ (P doesn't contain any vertex of S, except its ends i and j)

Definition 10 (Adjacent)**.**
i and $j \in R$ are *adjacent* with respect to a couple (x, S) if there exists an empty path (with respect to (x, S)) P between them.

See appendix C to get examples.

Definition 11 (Projection P_2 : from STSP to TSP)**.**

$$\begin{aligned} P_2 : \ \mathbb{R}^{|A|} &\rightarrow \mathbb{R}^{|R|^2} \\ x &\rightarrow x' \end{aligned} \tag{41}$$

Where $P_2(x)$ is defined by the following algorithm:

$T = R \cup \{0\}$ set of vertices to visit (the depot is counted twice).
$i = 0$ we begin the tour with the depot.
while $T \neq \emptyset$ **do**

Choose $j \in T$ such that i and j are *adjacent* in (x,T) (*)
$x'_{ij} = 0$

while $\exists P$ *an empty path (in* (x, T)*) between* i *and* j **do**

$m = \min_{(uv) \in P} x_{uv}$
$x'_{ij} \leftarrow x'_{ij} + m$
$x_{uv} \leftarrow x_{uv} - m : \ \forall (uv) \in P$

end
$T = T \setminus \{i\}$
$i = j$
end

Remark. The algorithm may be simplified by taking only the second `while` loop, but this detailed procedure helps to study the formulations and to make proofs.

Proof of P_2 correctness.
Assume that there is no sub-tour between a Steiner vertex and any other vertex (this hypothesis will be verified when using P_2).
Each arc (uv) which has $x(uv) > 0$ is part of at least a path between two required vertices, and then will be taken into account in the algorithm. It will be taken into account entirely because of the conservation of x at each vertex.

■

Proposition 2. *Let $x \in \mathbb{R}^{|A|}$ such that there is no sub-tour between a required vertex and a Steiner vertex and $x' = P_2(x)$. Then:*

(i)

$$x'_{ij} \leq 1 \ \forall i, j \in R \tag{42}$$

(ii)

$$\forall i \in R, \exists! j, k \in R / \ x'_{ij} > 0 \ \textit{and} \ x'_{ki} > 0 \tag{43}$$

For each vertex, there exists only one arc entering and one arc leaving it.

(iii)

$$d_{x'} = d_x$$

Proof.

(i) Let i and $j \in R$.
If i and j are chosen at line (*) of the algorithm, then
$x'_{ij} = \sum_{P \in P_{ij}} \min_{(uv) \in P} x_{uv} = \min_{\substack{S \subset V \\ i \in S, j \notin S}} x\left(S : \bar{S}\right) \leq x\left(\{i\} : V \setminus \{i\}\right) \leq 1$ by definition.
Otherwise, $x'_{ij} = 0$

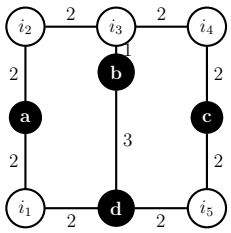
(ii) This is due to the fact that each vertex is taken only once in the projection P_2.

■

8.2.1 Examples

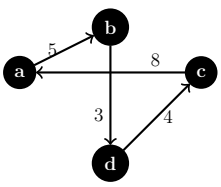
Example 1.

Given the following instance (d is the depot) with the distance matrix:



	a	b	c	d
a	0	5	8	4
b	5	0	5	3
c	8	5	0	4
d	4	3	4	0

Let the vector x' with the following non-zero variables: $x'_{dc} = 1$ $x'_{ca} = 1$ $x'_{ab} = 1$ $x'_{bd} = 1$:

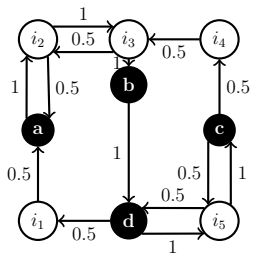


$d_{x'} = 20$.

We compute the shortest paths between each required vertices having $x' > 0$:

Pair of vertices	Shortest paths	
$(d; c)$	$(di_5)(i_5c)$	
$(c; a)$	$(ci_4)(i_4i_3)(i_3i_2)(i_2a)$	$(ci_5)(i_5d)(di_1)(i_1a)$
$(a; b)$	$(ai_2)(i_2i_3)(i_3b)$	
$(b; d)$	(bd)	

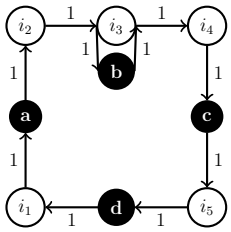
The projection P_1 gives:
$x_{i_1a} = \frac{1}{2}x'_{ca} = 0.5$ because the arc x_{i_1a} belongs to one shortest path between c and a out of 2 possibles paths.
$x_{i_2i_3} = x'_{ab} = 1$ because the arc $x_{i_2i_3}$ belongs to the unique shortest path between a and b
With the same reasoning we compute:
$x_{i_3i_2} = \frac{1}{2}x'_{ca} = 0.5$
$x_{ai_2} = x'_{ab} = 1$
$x_{i_2a} = \frac{1}{2}x'_{ca} = 0.5$
$x_{bd} = x'_{bd} = 1$
$x_{i_3b} = x'_{ab} = 1$
$x_{i_5c} = x'_{dc} = 1$
$x_{ci_5} = \frac{1}{2}x'_{ca} = 0.5$
$x_{ci_4} = \frac{1}{2}x'_{ca} = 0.5$
$x_{di_1} = \frac{1}{2}x'_{ca} = 0.5$
$x_{di_5} = x'_{dc} = 1$
$x_{i_5d} = \frac{1}{2}x'_{ca} = 0.5$
$x_{i_4i_3} = \frac{1}{2}x'_{ca} = 0.5$



$d_x = 20$

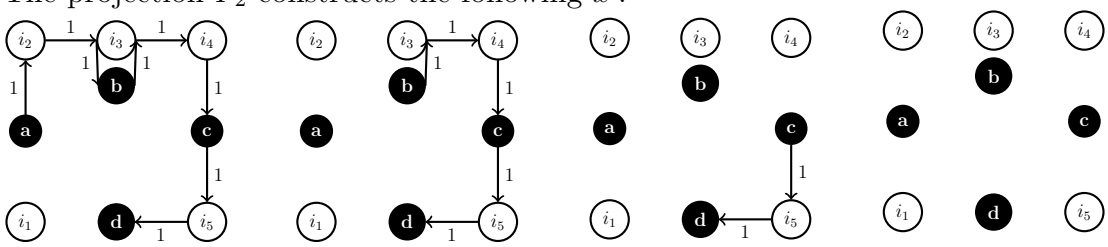
Example 2.

Let's consider the same data than in example 1 and the following value for x:

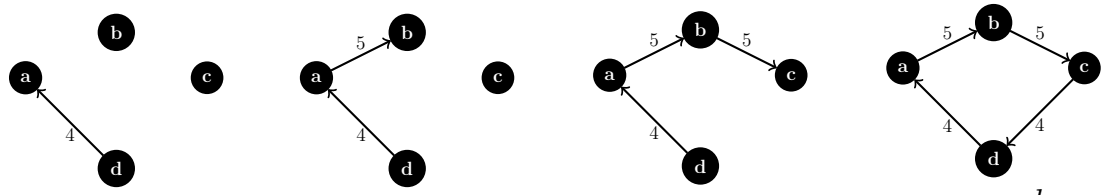


$d_x = 18.$

The projection P_2 constructs the following x':



$x'_{da} = 1$ $x'_{ab} = 1$ $x'_{bc} = 1$ $x'_{cd} = 1$



$d_{x'} = 18$

8.3 Theoretical results

Lemma 1. $SCFS_{LP} \leq SCF_{LP}$
The linear relaxation of the Steiner single commodity flow formulation is ***weaker*** *than the linear relaxation of the standard TSP single commodity flow formulation.*

Proof.

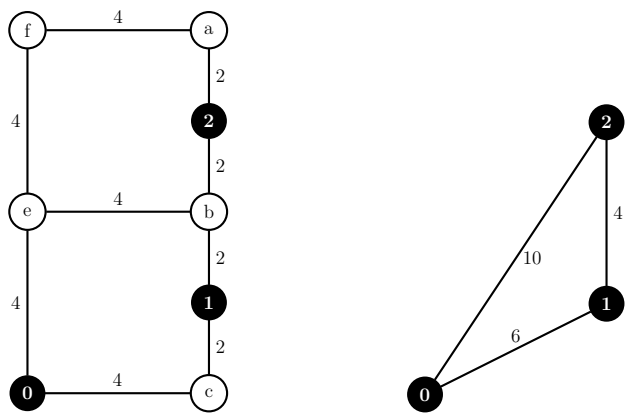
- $P^*_{SCF} \subset P^*_{SCFS}$

Let $(x', y') \in P_{SCF}$ an optimal fractional solution of SCF_{LP} and $(x, y) = (P_1(x'), P_1(y'))$. Then (x, y) is a feasible solution of P_{SCFS} since:
The assignment constraints (15) are respected due to the property (i) of P_1 (39).
The conservation constraints are respected due to the property (ii) of P_1 (40).
With the same hints, the flow constraints (17) and (18) are respected.
The bound on y (19) is respected: since the transformation is the same on x' and y' to obtain x and y, the ratio is kept: $\frac{y_{uv}}{x_{uv}} = \frac{y'_{ij}}{x'_{ij}}$ $\forall (uv) \in P_{ij}, \forall i, j \in R$

Thus, (x, y) is feasible for $SCFS$ and we have $z = z'$, this means $SCFS_{LR} \leq SCF_{LR}$

- $P^*_{SCFS} \not\subset P^*_{SCF}$

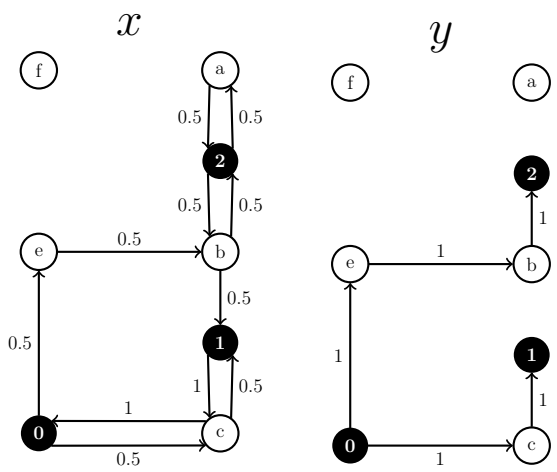
Consider the following example. We now show that $z^*_{LP} \leq 18 < z'^*_{LP} = 20$ so that $P^*_{SCFS} \not\subset P^*_{SCF}$:
To the left stands the Steiner representation of the example, to the right is the standard TSP representation (with a complete graph).



The resolution of a standard TSP (in the complete graph) by the single-commodity flow formulation leads to:

Objective of SCF_{LP}: $\quad min\ 6(x'_{01} + x'_{10}) + 10(x'_{02} + x'_{20}) + 4(x'_{12} + x'_{21})$
Constraints (8) and (9) impose :
$x'_{01} + x'_{02} = 1$
$x'_{10} + x'_{20} = 1$
$x'_{21} + x'_{20} = 1$
$x'_{12} + x'_{02} = 1$
$x'_{12} + x'_{10} = 1$
$x'_{21} + x'_{01} = 1$
So we have :
${z'_{LP}}^* = 4(x'_{02} + x'_{12}) + 6(x'_{02} + x'_{01}) + 4(x'_{20} + x'_{21}) + 6(x'_{20} + x'_{10}) = 20$

On the other side, we can easily build a feasible solution of SCFS_{LP} with a cost 18 as follows :



Thus, in this example we have $z_{LP}{}^* \leq d_x < z'_{LP}{}^*$

■

Lemma 2. $DFJS_{LR} = DFJ_{LR}$
The linear relaxations of the sub-tours cuts formulation are ***equivalent*** *in Steiner and in standard TSP approaches.*

Proof.

- $P_{DFJ} \subseteq P_{DFJS}$.

Let $x' \in P_{DFJ}$ and $x = P_1(x')$. Then:
Assignment constraints (24) are respected immediately from (40).
Constraints of sub-tour elimination (25) are respected:
By definition, the transformation states: $x\left(S : \bar{S}\right) \geq x'\left(S' : \bar{S'}\right)\ \forall S \in V, S' = S \cap R$.
Assume $\exists S \subset V$ such that $S \cap R = S' \neq \emptyset$, $R \setminus S = \bar{S'} \neq \emptyset$ and $x\left(S : \bar{S}\right) < 1$
Thus, $x'\left(S' : \bar{S'}\right) < 1$ which contradicts feasibility of x'.

Thus, $x \in P_{DFJS}$.

- $P_{DFJS} \subseteq P_{DFJ}$.

Let $x \in P_{DFJS}$ and $x' = P_2(x)$. By definition of x, there is no sub-tour between a Steiner vertex and any other vertex and the properties of P_2 hold. Then, x' is an optimal solution of DFJ_{LP}:
The sub-tour elimination constraints (4) are obviously respected: from the similar constraints in Steiner model (25), we know that $x\left(S : \bar{S}\right) \geq 1 \forall S \subset V : S \cap R \neq \emptyset, R \setminus S \neq \emptyset$. Yet, we can define $S_R = S \cap R$ and then $x'\left(S_R : \bar{S_R}\right) \geq 1$.
The assignment constraints are respected due to (42) and (43) .
Thus, $x' \in P_{DFJ}$.

■

9 Experimental Results

9.1 Implementation

We used the CPLEX Java API (version 12.6) to solve the different linear programs. An useful function of this API is the "warm start". It allows us to give CPLEX a starting feasible integer solution. This adds an upper bound and then improve the search tree.

A way to get a starting feasible integer solution is to apply a heuristic. We used the software LKH (freely available at `http://webhotel4.ruc.dk/~keld/research/LKH/`) which is an effective implementation of the Lin-Kernighan heuristic [23]. This heuristic, dedicated to the TSP, improves iteratively a solution by swapping pairs of sub-tours to make a new tour.
It is a generalization of the heuristic "k-opt" (which is itself a generalization of 2-opt) in which, at each step, the algorithm switch k paths to make the tour shorter. The Lin-Kernighan is more efficient because it is adaptive: at each step it determines how many paths must be switched to improve the solution [28].

The academic benchmark solved was proposed by Theys, Bräysy, Dullaert and Raa[39]. Computational testing was done on an Intel Xeon E5-2440 v2 @ 1.9 GHz processor and 32 GB of RAM. The experiments ran with a memory limit of 8 GB of RAM.

9.2 Classes of instances

A class of instances is described and named by its number of aisles, cross-aisles and products and by the storage policy in the warehouse. We focus on a subset of the benchmark and study only the following 12 classes of instances:

Class	Number of aisles	Number of cross-aisles	Number of products
15_3_60	15	3	60
15_6_15	15	6	15
15_6_240	15	6	240
15_6_60	15	6	60
60_11_240	60	11	240
60_3_240	60	3	240

Each one is considered with a random (_Random version) and a volume (_Volume version) distribution of products. In the random version, the products are stored randomly in the warehouse while in the volume version 80% of the products are gathered in close sub-aisles. We solve 10 instances of each class.

9.3 Solvers

We compare the performances of the following algorithms:

- SCFS0: the basic Steiner single commodity flow formulation
- SCFS1: SCFS0 + preprocessing
- SCFS2: SCFS1 + sub-tour cuts
- SCFS+: SCFS2 + dominance
- SCF+: the standard single commodity flow formulation with vertex preprocessing

Class	TSP	TSP+	Evolution	Steiner	Steiner TSP+	Evolution
15_3_60_Random	3660	3266	-11%	246	233	-5%
15_3_60_Volume	3660	2425	-34%	210	158	-25%
15_6_15_Random	240	234	-3%	351	172	-51%
15_6_15_Volume	240	237	-1%	350	124	-65%
15_6_240_Random	57840	45336	-22%	630	623	-1%
15_6_240_Volume	57840	26280	-55%	493	459	-7%
15_6_60_Random	3660	3470	-5%	434	381	-12%
15_6_60_Volume	3660	3049	-17%	409	285	-30%
60_11_240_Random	57840	47150	-18%	2935	2512	-14%
60_11_240_Volume	57840	26377	-54%	2826	2044	-28%
60_3_240_Random	57840	47150	-18%	968	934	-4%
60_3_240_Volume	57840	26571	-54%	786	645	-18%

Table 2: Average number of arcs in TSP graph and Steiner grap, with and without preprocessing.

- CDE: Concorde
- CDE+: concorde with preprocessed input
- PDYN: dynamic programming [7]

9.4 Results

In this section we present some results that shows the single-commodity flow formulation with the different improvements provided in our work is an efficient algorithm to solve the academic benchmark described above. This part will be divided into different paragraphs. First, we make an analysis of the size, which is the biggest advantage of the SCFS formulation. Then we analyze the strength of each formulation and note that the linear relaxation of SCFS strongly benefits from the improvements proposed. Following this, we compare resolution times. We conclude by noticing that the preprocessing on Concorde is really efficient.

Size analysis

Table 2 shows the number of arcs in the TSP case (complete graph) with and without the "Concorde" preprocessing (see paragraph in section 7.1.1) and in the Steiner graph with and without the complete preprocessing (see section 7.1). The columns "Evolution" show the percentage of arcs removed when the preprocessing is applied. We notice that in the case of TSP, even if the preprocessing suppresses less vertices than in SCFS, the saving is really important: it can reach more than 50%, due to the completeness of the graph.
We also observe that the benefits are higher when the warehouse has a *volume* policy which is an expected result given the proposed preprocessing.

The main observation is that the number of arcs in the Steiner graph is much smaller than in the complete graph, and thus the number of variables is also really small in our single-commodity flow formulation.

Analysis of lower bounds

Table 3 compares the average gap between the **linear relaxation** (LR) and the optimal value (OPT) for each class and for all Steiner single-commodity flow formulations. The gap

Class	SCFS0	SCFS1	SCFS2	SCFS+
15_3_60_Random	33,0%	28,9%	3,7%	1,3%
15_3_60_Volume	35,0%	24,6%	4,4%	0,8%
15_6_15_Random	36,3%	34,3%	9,6%	0,7%
15_6_15_Volume	39,0%	37,0%	3,2%	0,4%
15_6_240_Random	40,9%	30,9%	3,7%	1,5%
15_6_240_Volume	47,3%	30,1%	10,3%	1,8%
15_6_60_Random	36,7%	35,0%	13,0%	1,4%
15_6_60_Volume	42,1%	37,7%	8,4%	1,0%
60_11_240_Random	38,8%	33,2%	16,9%	2,6%
60_11_240_Volume	54,9%	30,6%	9,7%	1,1%
60_3_240_Random	42,1%	37,4%	1,0%	0,5%
60_3_240_Volume	54,7%	35,9%	1,2%	0,3%
Total	41,7%	33,0%	7,1%	1,1%

Table 3: Average gap of linear relaxation to optimal value for Steiner single-commodity flow formulations

is computed as $\frac{OPT-LR}{OPT} \times 100$. This shows that the different improvements made on this formulation are really effective. The gap moves from 42% to 1% in average without increasing significantly the computing time which moves from 0.1 second to 0.4 second in average.
Thus, this result proves that our work on the SCFS formulation is efficient, and each improvement has a significant impact on the linear relaxation.

To compare our results to Concorde, we observe the gap between the **value at the root node** of the tree search (which is also a lower bound) and the optimal. This value is better than the linear relaxation since CPLEX and Concorde apply many treatments to improve it. Table 4 compares this gap in the different formulations.

Class	SCFS+	SCF+	CDE+
15_3_60_Random	0,33%	2,87%	0,00%
15_3_60_Volume	0,20%	1,44%	0,00%
15_6_15_Random	0,00%	0,42%	0,00%
15_6_15_Volume	0,04%	0,70%	0,04%
15_6_240_Random	0,83%	2,90%	0,28%
15_6_240_Volume	0,74%	1,61%	0,02%
15_6_60_Random	0,44%	1,93%	0,03%
15_6_60_Volume	0,04%	2,60%	0,00%
60_11_240_Random	1,85%	1,95%	0,03%
60_11_240_Volume	0,65%	2,90%	0,00%
60_3_240_Random	0,35%	10,26%	0,02%
60_3_240_Volume	0,05%	15,55%	0,07%
Total	0,46%	3,76%	0,04%

Table 4: Average gap between value at root node and optimal value

These different observations show that the SCFS+ has a good quality despite the fact that the initial formulation SCFS is weaker than all the other ones. In practice, we observe that $SCFS^{+}_{LP} \geq SCF_{LP}$. Namely, the linear relaxation of the **improved** Steiner single commodity flow formulation is **stronger** than the linear relaxation of standard single commodity flow formulation.

Performances

In practice we limited the resolution time to 30 minutes for SCFS+ and SCF+ resolution. Some instances were unsolved in this time limit. Table 5 shows the number of unsolved instances after 30 minutes of processing.

SCFS+	10
SCF+	43
CDE	1
CDE+	0
PDYN	10 (memory issue)

Table 5: Number of unsolved instances after 30 minutes

We can notice that the improved Steiner formulation proposed in this report completely outperforms the standard compact TSP formulation. Note that the dynamic programming will never be able to solve instances with a lot of horizontal cross-aisles due to memory issues, while we can hope that further work on SCFS will enable it to overcome current limitation.
Moreover, the instances that cannot be solved in 30 minutes by SCFS+ are from classes 60_ 11_ 240_ Random and 60_3_240_Random, which are big instances (the warehouse is big and there are many products to collect) and, above all, they are *random* instances, which means that the products are dispersed in the warehouse. Actually, we observe that the results on *volume* instances are really better. This suggests that optimizing the storage policy of the warehouse is important.

To compare resolution time, we choose the instances solved in less than 30 minutes by all solvers SCFS+, CDE+ and PDYN. Formulation SCF+ has too much unsolved instances after this time limit so it appears irrelevant to include it in the comparison. Table 6 shows the numerical results. In average, the SCFS+ is slower than Concorde and the dynamic programming. However, on many instances the computing time is reasonable.

Class	**SCFS+**	**CDE+**	**PDYN**
15_3_60_Random	0,64	0,11	0,00
15_3_60_Volume	0,19	0,05	0,00
15_6_15_Random	0,12	0,00	0,29
15_6_15_Volume	0,09	0,01	0,28
15_6_240_Random	459,30	9,77	0,40
15_6_240_Volume	10,15	0,35	0,33
15_6_60_Random	2,52	0,07	0,31
15_6_60_Volume	0,49	0,04	0,32
60_3_240_Random	27,30	1,18	0,01
60_3_240_Volume	10,80	1,95	0,01
Sub total	51,16	1,35	0,19
60_11_240_Random	486,63	0,34	-
60_11_240_Volume	125,31	0,22	-
Total	62,08	1,24	-

Table 6: Average time of optimal resolution (in seconds) for instances solved in less than 30 minutes with each solver

Optimal resolution

The preprocessing on Concorde improved the resolution time. First, an instance which was solved in 18069 seconds before, is now solved in 13 seconds.
Moreover, the average time of resolution moved from 2.5 seconds (without taking in account the instance solved in 180069 seconds) to 1.19 seconds (note that this figure is different from the total in Table 6 since we add the 10 instances that SCFS does not solve).
Moreover, any instance of the entire benchmark of [39](only a subset was considered above) can be in fact solved optimally in less than 5 minutes by Concorde.

10 Conclusion and future work

This research project aimed to find new exact approaches to solve the picking problem. A contribution was made on several sides. In particular, preprocessing on data was found, available for different approaches and a mixed-integer linear program was proposed, with experimental results very promising for real application.

We have studied compact MILP formulations for the TSP that can easily accommodate side constraints and be embedded in warehouse management systems (WMS) unlike dedicated approaches such as Concorde or dynamic programming. We showed that, on one hand, the compact formulations based on modeling the problem as a TSP do not scale in memory and are unable to solve realistic size instances. On the other hand, a flow based formulation modeling the problem as a Steiner TSP is very sparse but we proved that it has a weaker linear relaxation. As a result, it is also very inefficient in practice.
We thus proposed a number of improvements by taking advantage of the warehouse structure. These improvements are based on cutting planes, dominances and procedures to significantly reduce the instance's size without loosing optimality. The resulting model remains sparse and exhibits a very strong linear relaxation in practice. It outperforms the compact TSP model and proves able to solve very large instances efficiently up to being nearly competitive with dedicated TSP approaches on the benchmark studied.

Note finally that some of the ideas proposed here can be applied to improve the efficiency of Concorde. The entire benchmark proposed by Theys, Bräysy, Dullaert and Raa[39] can be thus be solved to optimality very efficiently without the need of the heuristics proposed by the same authors.

Many tracks for further work are relevant in two different directions: specialization or generalization.
In the first case, the purpose would be to get stronger reasoning on the structure of the data to get more efficient algorithm. It would be moreover desirable to obtain an industrial benchmark to test our algorithm on real instances.
It may be also interesting to link this problem with other warehouse issues. Some heuristics were proposed to solve the joint problem of batching and routing in warehouse [41]. We also observed that storage policy has an important impact on the resolution complexity of the picker routing problem, and a joint study could be an promising track.
In the second case, the idea is to generalize or adapt the work done here to the case of any Steiner TSP and find other fields of application. This structure of rectangular grid could be applied to the TSP in a city such as Manhattan or industrial application like microelectronics manufacturing.

Bibliography

[1] David Applegate, Robert Bixby, Vašek Chvátal, and William Cook. Concorde. Available at: `http://www.math.uwaterloo.ca/tsp/`.

[2] David Applegate, Robert Bixby, Vašek Chvátal, and William Cook. Implementing the dantzig-fulkerson-johnson algorithm for large traveling salesman problems. *Mathematical programming*, 97(1-2):91–153, 2003.

[3] David L Applegate, Robert E Bixby, Vašek Chvátal, William Cook, Daniel G Espinoza, Marcos Goycoolea, and Keld Helsgaun. Certification of an optimal tsp tour through 85,900 cities. *Operations Research Letters*, 37(1):11–15, 2009.

[4] Marc Benkert, Alexander Wolff, Florian Widmann, and Takeshi Shirabe. The minimum manhattan network problem: approximations and exact solutions. *Computational Geometry*, 35(3):188–208, 2006.

[5] Sylvia C Boyd and William H Cunningham. Small travelling salesman polytopes. *Mathematics of Operations Research*, 16(2):259–271, 1991.

[6] Rainer E Burkard, Vladimir G Deineko, René van Dal, Jack AA van der Veen, and Gerhard J Woeginger. Well-solvable special cases of the traveling salesman problem: a survey. *SIAM review*, 40(3):496–546, 1998.

[7] Hadrien Cambazard and Nicolas Catusse. Fixed-parameter algorithms for rectilinear steiner tree and rectilinear traveling salesman problem in the plane. *arXiv preprint arXiv:1512.06649*, 2015.

[8] Francis YL Chin, Zeyu Guo, and He Sun. Minimum manhattan network is np-complete. *Discrete & Computational Geometry*, 45(4):701–722, 2011.

[9] Václav Chvátal. Edmonds polytopes and weakly hamiltonian graphs. *Mathematical programming*, 5(1):29–40, 1973.

[10] Gianni Codato and Matteo Fischetti. Combinatorial benders' cuts for mixed-integer linear programming. *Operations Research*, 54(4):756–766, 2006.

[11] Gérard Cornuéjols, Jean Fonlupt, and Denis Naddef. The traveling salesman problem on a graph and some related integer polyhedra. *Mathematical programming*, 33(1):1–27, 1985.

[12] George Dantzig, Ray Fulkerson, and Selmer Johnson. Solution of a large-scale traveling-salesman problem. *Journal of the operations research society of America*, 2(4):393–410, 1954.

[13] René De Koster, Tho Le-Duc, and Kees Jan Roodbergen. Design and control of warehouse order picking: A literature review. *European Journal of Operational Research*, 182(2):481–501, 2007.

[14] René De Koster and Edo Van der Poort. Routing orderpickers in a warehouse: a comparison between optimal and heuristic solutions. *IIE transactions*, 30(5):469–480, 1998.

[15] Bernhard Fleischmann. A cutting plane procedure for the travelling salesman problem on road networks. *European Journal of Operational Research*, 21(3):307–317, 1985.

[16] Bernhard Fleischmann. A new class of cutting planes for the symmetric travelling salesman problem. *Mathematical Programming*, 40(1-3):225–246, 1988.

[17] Bezalel Gavish and Stephen C Graves. The travelling salesman problem and related problems. 1978.

[18] Ralph E. Gomory. Outline of an algorithm for integer solutions to linear programs. 1958.

[19] Martin Grötschel and Manfred W Padberg. On the symmetric travelling salesman problem i: inequalities. *Mathematical Programming*, 16(1):265–280, 1979.

[20] Martin Grötschel and William R Pulleyblank. Clique tree inequalities and the symmetric travelling salesman problem. *Mathematics of operations research*, 11(4):537–569, 1986.

[21] Michael Hahsler and Kurt Hornik. Tsp infrastructure for the traveling salesperson problem. *Journal of Statistical Software*, 23(1):1–21, 2007.

[22] Randolph W Hall. Distance approximations for routing manual pickers in a warehouse. *IIE transactions*, 25(4):76–87, 1993.

[23] Keld Helsgaun. An effective implementation of the lin–kernighan traveling salesman heuristic. *European Journal of Operational Research*, 126(1):106–130, 2000.

[24] Alon Itai, Christos H Papadimitriou, and Jayme Luiz Szwarcfiter. Hamilton paths in grid graphs. *SIAM Journal on Computing*, 11(4):676–686, 1982.

[25] Richard M Karp. *Reducibility among combinatorial problems*. Springer, 1972.

[26] Adam N Letchford. Separating a superclass of comb inequalities in planar graphs. *Mathematics of Operations Research*, 25(3):443–454, 2000.

[27] Adam N Letchford, Saeideh D Nasiri, and Dirk Oliver Theis. Compact formulations of the steiner traveling salesman problem and related problems. *European Journal of Operational Research*, 228(1):83–92, 2013.

[28] Shen Lin and Brian W Kernighan. An effective heuristic algorithm for the traveling-salesman problem. *Operations research*, 21(2):498–516, 1973.

[29] François Margot. Symmetry in integer linear programming. In *50 Years of Integer Programming 1958-2008*, pages 647–686. Springer, 2010.

[30] Samuel A Mulder and Donald C Wunsch. Million city traveling salesman problem solution by divide and conquer clustering with adaptive resonance neural networks. *Neural Networks*, 16(5):827–832, 2003.

[31] Denis Naddef and Giovanni Rinaldi. The symmetric traveling salesman polytope: New facets from the graphical relaxation. *Mathematics of Operations Research*, 32(1):233–256, 2007.

[32] CS Orloff. A fundamental problem in vehicle routing. *Networks*, 4(1):35–64, 1974.

[33] AJ Orman and H Paul Williams. A survey of different integer programming formulations of the travelling salesman problem. *Optimisation, economics and financial analysis. Advances in computational management science*, 9:93–106, 2006.

[34] Charles G Petersen. An evaluation of order picking routeing policies. *International Journal of Operations & Production Management*, 17(11):1098–1111, 1997.

[35] Charles G Petersen and Gerald Aase. A comparison of picking, storage, and routing policies in manual order picking. *International Journal of Production Economics*, 92(1):11–19, 2004.

[36] H Donald Ratliff and Arnon S Rosenthal. Order-picking in a rectangular warehouse: a solvable case of the traveling salesman problem. *Operations Research*, 31(3):507–521, 1983.

[37] Kees Jan Roodbergen and René De Koster. Routing methods for warehouses with multiple cross aisles. *International Journal of Production Research*, 39(9):1865–1883, 2001.

[38] Kees Jan Roodbergen and René De Koster. Routing order pickers in a warehouse with a middle aisle. *European Journal of Operational Research*, 133(1):32–43, 2001.

[39] Christophe Theys, Olli Bräysy, Wout Dullaert, and Birger Raa. Using a tsp heuristic for routing order pickers in warehouses. *European Journal of Operational Research*, 200(3):755–763, 2010.

[40] James A Tompkins, John A White, Yavuz A Bozer, and Jose Mario Azaña Tanchoco. *Facilities planning*. John Wiley & Sons, 2010.

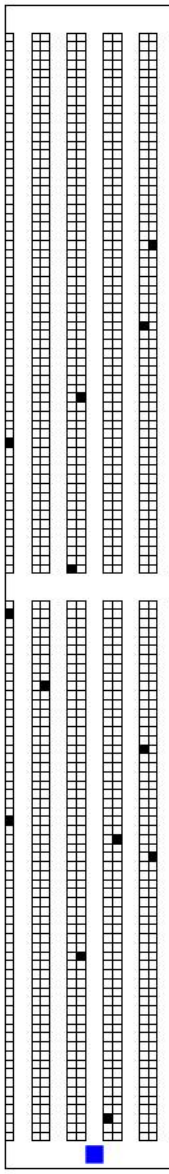
[41] J Won and S Olafsson*. Joint order batching and order picking in warehouse operations. *International Journal of Production Research*, 43(7):1427–1442, 2005.

Appendices

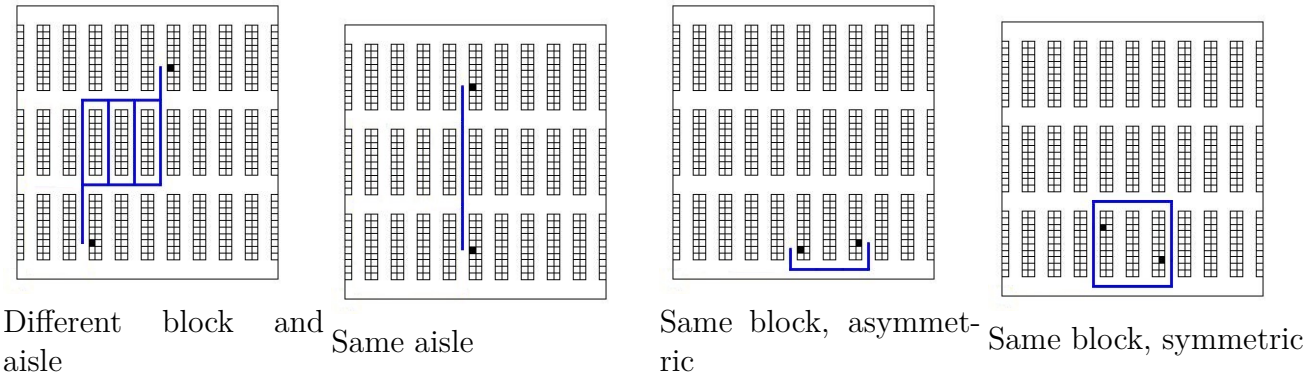
List of appendices

Appendix A: Instance file

```
nbAisles = 5;
nbCrossAisles = 3;
nbOrders = 15;
nrPlacesPerRack = 60;
aisleWidth = 2;
rackDepth = 2;
locationWidth = 1;
crossAisleWidth = 3;

warehouseX = [6 4 8 0 8 5 2 0 9 0 7 7 3 4 9 ];

warehouseY = [86 99 88 84 23 117 69 61 102 45 32 76 59 40 73 ];

locationsX = [10 14 10 18 2 18 10 6 2 18 2 14 14 6 10 18 ];

locationsY = [127.5 92.5 105.5 94.5 90.5 26.5 123.5 75.5 67.5 108.5 48.5 35.5 82.5

distances = [
[0 39 22 41 45 109 4 56 68 27 87 96 49 69 84 56 ]
[39 0 61 62 66 70 43 47 43 58 56 57 10 38 53 47 ]
[22 61 0 63 67 87 18 56 52 49 65 74 63 47 62 64 ]
[41 62 63 0 72 68 45 53 49 14 62 63 52 44 59 15 ]
[45 66 67 72 0 80 49 41 23 72 42 67 56 32 55 57 ]
[109 70 87 68 80 0 105 61 57 82 70 63 60 52 67 53 ]
[4 43 18 45 49 105 0 60 70 31 83 92 53 65 80 60 ]
[56 47 56 53 41 61 60 0 18 67 31 48 37 13 36 38 ]
[68 43 52 49 23 57 70 18 0 63 19 44 33 9 32 34 ]
[27 58 49 14 72 82 31 67 63 0 76 77 66 58 73 29 ]
[87 56 65 62 42 70 83 31 19 76 0 57 46 22 45 47 ]
[96 57 74 63 67 63 92 48 44 77 57 0 47 39 54 48 ]
[49 10 63 52 56 60 53 37 33 66 46 47 0 28 43 37 ]
[69 38 47 44 32 52 65 13 9 58 22 39 28 0 27 29 ]
[84 53 62 59 55 67 80 36 32 73 45 54 43 27 0 44 ]
[56 47 64 15 57 53 60 38 34 29 47 48 37 29 44 0 ]
];
```



Appendix B: Intervals



Appendix C: Adjacency

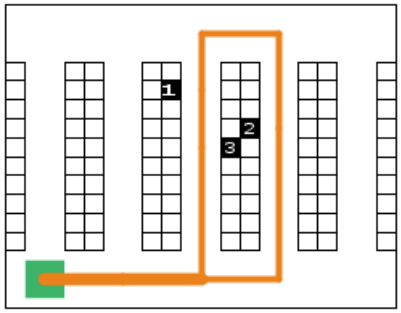


Figure 15: The lines represent a vector x. Let $S = \{depot, 1, 2, 3\}$. 2 and 3 are not adjacent while 1 and 2 are adjacent with respect to (x, S).