# Exact algorithms for the order picking problem

Lucie Pansart, Nicolas Catusse, Hadrien Cambazard

**To cite this version:**

Lucie Pansart, Nicolas Catusse, Hadrien Cambazard. Exact algorithms for the order picking problem. Computers and Operations Research, 2018, 100, pp.117 - 127. 10.1016/j.cor.2018.07.002 . hal-01481915v4

**HAL Id: hal-01481915**
**https://hal.science/hal-01481915v4**

Submitted on 24 May 2018

**HAL** is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

# Exact algorithms for the order picking problem

Lucie Pansart[a,*], Nicolas Catusse[a], Hadrien Cambazard[a]

[a] *Univ.Grenoble Alpes, CNRS, G-SCOP, 38000 Grenoble, France*

**Abstract**

Order picking is the problem of collecting a set of products in a warehouse in a minimum amount of time. It is currently a major bottleneck in supply-chain because of its cost in time and labor force. This article presents two exact and effective algorithms for this problem. Firstly, a sparse formulation in mixed-integer programming is strengthened by preprocessing and valid inequalities. Secondly, a dynamic programming approach generalizing known algorithms for two or three cross-aisles is proposed and evaluated experimentally. Performances of these algorithms are reported and compared with the Traveling Salesman Problem (TSP) solver Concorde.

*Keywords:* integer programming, order picking, Steiner TSP, TSP, dynamic programming

## 1. Introduction

The order picking activity lies at the heart of logistic. It 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 since it is estimated that it accounts for 55% of the total operational warehouse costs [26]. Many order picking systems are currently used in warehouses. The methods are classified following who picks the orders (humans or machines), who moves (picker or products) and the strategy used. We focus here on manual systems where the order picker moves in a regular rectangular warehouse.

An efficient way to optimize order picking in this case is to reduce picker travel time. Thus, we are concerned with the following issue: **how to optimize the routing time in the warehouse?** This problem is called order picking problem, picker routing problem, or picking problem for sake of simplicity. To the best of our knowledge, this problem cannot be solved in polynomial time, except when the number of cross-aisles is bounded [4, 21].

*Corresponding author

*Email addresses:* `lucie.pansart@g-scop.grenoble-inp.fr` (Lucie Pansart), `nicolas.catusse@g-scop.grenoble-inp.fr` (Nicolas Catusse), `hadrien.cambazard@g-scop.grenoble-inp.fr` (Hadrien Cambazard)

We show in this paper that the problem can be efficiently solved optimally with mixed integer programming using a sparse formulation as well as adequate preprocessing and valid inequalities. Moreover, we extend a dynamic programming algorithm initially proposed by Ratliff and Rosenthal for 2 cross-aisles [20], to any number of cross-aisles and evaluate it experimentally. It turns out to scale for a number of cross-aisles large enough to deal with real-life warehouses. This approach is however less suited to accommodate side-constraints.

This article begins with a description of the picking problem and a literature review (section 2). Section 3 presents different ways of preprocessing the data to improve algorithms efficiency. Two exact algorithms are then presented in sections 4 and 5. Finally, we report experimental results in section 6.

## 2. Problem description and literature review

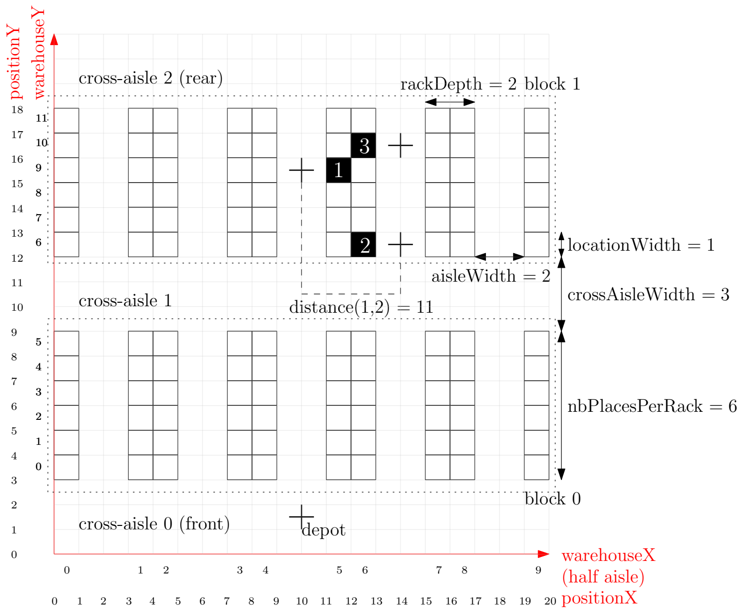
We consider a regular rectangular warehouse with a single depot used to take the order and to drop it off. The warehouse is made of $v$ **vertical aisles** and $h$ **horizontal cross-aisles**. Products are located on both sides of vertical aisles. Cross-aisles do not contain any products but enable the order picker to navigate in the warehouse (see Figure 1(a)). We make two common hypothesis on the warehouse:

**Hypothesis 1.** *Aisles are narrow (the distance for crossing is considered null).*

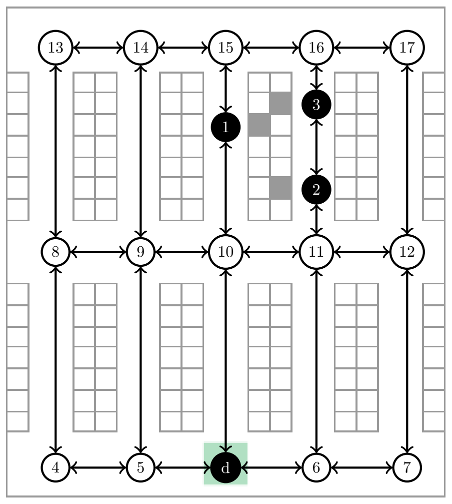
**Hypothesis 2.** *All aisle's lengths are equal. All cross-aisle's lengths are equal.*

An **order** is given by a picking list, i.e. a set of $n$ products, indexed from 1 to $n$ and described by their location in the warehouse. We define by $\mathbf{R}$ the indexes of all locations to visit (the products) including 0 which is the index of the depot so $R = \{0, \ldots, n\}$. The set $\mathbf{I}$ denotes the indexes of intersections between aisles and cross-aisles of the warehouse, excluding the depot. The set of all relevant locations in the warehouse is therefore $\mathbf{V} = I \cup R$. 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), the well-known $\mathcal{NP}$-hard [14] problem, where the salesman is the order picker and cities are products to collect. This problem, introduced by Dantzig, Fulkerson and Johnson in 1954 [7], is one of the most studied in Operations Research. The survey of Orman and Williams [18] gives an overview of integer programming formulations for the TSP. Efficient exact algorithms have been designed for the TSP and Concorde is one of the best exact solver (see Hahsler and Hornik [11] or Mulder and Wunsch [16]). To solve the specific case of picking problem, many heuristics have been proposed in particular by Hall [12]. Some performance analysis of the most popular heuristics were made by Petersen [19] and by Roodbergen and De Koster [22]. Theys, Bräysy, Dullaert and Raa [24] proposed to combine classical TSP heuristics with picking heuristics and provided a benchmark which is used in this work.



(a) Layout description of the considered warehouse



(b) Graph representation of the considered warehouse

Figure 1: Example of a warehouse. A representation of the different parameters of a layout and the corresponding Steiner graph are overlaid on the warehouse structure.

An exact approach using dynamic programming has been proposed for the first time by Ratliff and Rosenthal in the case of a single block (i.e., 2 cross-aisles) in 1983 [20]. It was extended by Roodbergen and De Koster [21] in the case of three cross-aisles. These algorithms are polynomial in the number of aisles and products of the warehouse and used in heuristics for cases with more than three cross-aisles [22, 28]. Cambazard and Catusse developed a dynamic programming approach which can solve any rectilinear TSP [4]. This algorithm can be applied to the picking problem for any rectangular warehouse with $h$ cross-aisles, but it is exponential in $h$. Although this extension was suggested in several articles [20, 21], it has never been detailed nor implemented. The reason is that the extensive analysis made by [20, 21] of the possible states of the dynamic program does not easily generalize beyond three cross-aisles. We give in section 5 a simple and general version of this exact algorithm for any number of cross-aisles to use it as a baseline in the experiments.

Practical comparisons have been performed between heuristics and exact algorithms, notably by De Koster and Van der Poort [8]. They compared the dynamic program with the commonly used S-shape heuristic and conclude 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 motivation for exact and efficient algorithms.

The problem can be seen as a Steiner TSP [23] which is a variant of the TSP proposed independently by Fleischmann [9] and Cornuéjols, Fonlupt and Naddef in 1985 [6] although Orloff introduced the idea some years before [17]. Burkard *et al.* [3] categorized the picking problem applied to the case of series-parallel graph in well-solvable special cases of the TSP, which includes warehouses with 2 cross-aisles [6]. Our Mixed Integer Linear Programming (MILP) model is based on the compact MILP formulations proposed by Letchord, Nasiri and Theis [15].

The Steiner TSP (also known as subset TSP) is stated in a directed graph $\mathbf{D} = (\mathbf{V}, \mathbf{A})$ (see Figure 1(b)) where $V$ is the set of vertices and $A$ is the set of arcs, defined as follows: $\forall i, j \in V,\ (ij) \in A \Leftrightarrow$ one of the following conditions holds:

1. $i$ and $j$ are horizontally adjacent intersections (e.g. 4 and 5 in Figure 1(b)).
2. $i$ and $j$ are extreme intersections of an empty sub-aisle (e.g. 4 and 8).
3. $j$ is an extreme products and $i$ the adjacent intersection (e.g. 3 and 16).
4. $i$ and $j$ are adjacent products (e.g. 2 and 3).

Each arc $(ij)$ is weighted by $d_{ij}$, the distance between $i$ and $j$ in the warehouse. We study here the symmetric TSP, which means the graph D is such that $\forall (ij) \in A : (ji) \in A$ and $d_{ij} = d_{ji}$. We generalize this distance by defining the function $d : V \times V \to \mathbb{N}$ as follows: $d(i, j)$ is the distance of a shortest path between vertices $i$ and $j$.

We define by $P_{ij}$ the set of all shortest paths between the two products $i$ and $j$. More formally, $P_{ij} = \{P = (v_0 v_1 v_2 ... v_K) | v_0 = i,\ v_K = j,\ (v_{k-1} v_k) \in A$ $\forall k = 1...K$ and $\sum\limits_{k=1}^{K} d_{v_{k-1} v_k} = d(i, j)\}$. The set of **neighbors** of a vertex $i$ is denoted $\Gamma(i)$ so that $\Gamma(i) = \{j \in V | (ij) \in A\}$. Note that we don't need to distinguish the successeurs and predecessors of a vertex since they are the same by construction of $D$.

## 3. Preprocessing

The size of the problem can be considerably reduced, which is a good lever to reduce computation time. First, we can reduce the size of the picking list, i.e., the number of vertices in $R$. Then, we show how to reduce the number of arcs in the graph, keeping a sufficient set of arcs to find an optimal solution. Figure 2 shows an example of the impact of the preprocessing.

### *3.1. Reducing the number of vertices*

In this part, we describe two preprocessings reducing the number of products without changing the problem. The first algorithm must be used only when the solver accepts additional constraints while the second one is general.

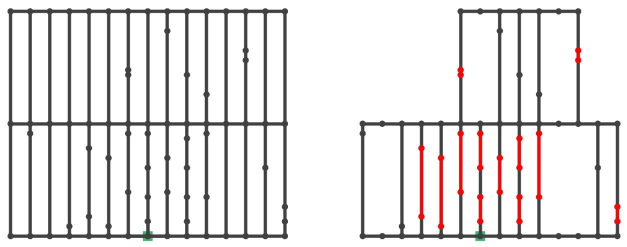


Figure 2: Example of an instance and its preprocessing. Red edges show edges forced by the node preprocessing. The instance has 105 vertices and 270 arcs while the preprocessed instance has 65 vertices and 156 arcs.

Both algorithms are based on the following result due to Ratliff and Rosenthal: in an optimal solution, there exists six unique ways to traverse a sub-aisle (shown in Figure 3) [20]. Case (e) is the only configuration that might not be unique. However, only configurations where the gap between black and white groups is the largest will occur in an optimal solution. The largest gap of a sub-aisle is defined as the longest empty distance between two vertices of a sub-aisle. Several configurations can still exist if the largest gap is not unique in the sub-aisle, but they will all have the same cost in an optimal solution.

### *3.1.1. Additional constraints available*

In any case, the black vertices are all picked in one go (one after the other) 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 extreme products.

**Definition 1** (Preprocessing)**.** *The vertex preprocessing is defined by the following algorithm:*

```
for every sub-aisle do
    - Compute a largest gap between two vertices
    - 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 $t_S$ and $b_S$ (resp. $t_T$ and $b_T$) that are the farthest apart
    - Add the constraints forcing the order picker to traverse each set at least once. This means the order picker must traverse arc $(t_S b_S)$ or $(b_S t_S)$ and similarly arc $(t_T b_T)$ or $(b_T t_T)$. [1]
end
```

[1]In the MILP (see section 4) it translates into the following constraints:

$$x_{t_S b_S} + x_{b_S t_S} \geq 1 \text{ if } t_S \neq b_S \text{ and } x_{t_T b_T} + x_{b_T t_T} \geq 1 \text{ if } t_T \neq b_T$$

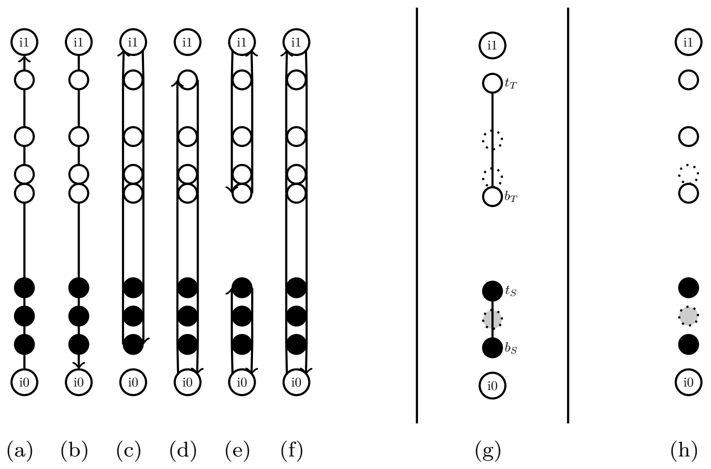


Figure 3: To the left: The six ways to traverse a sub-aisle. We distinguish two groups (black and white vertices) where all vertices of one group are taken in one go. In the middle: the result after preprocessing 3.1.1. Dashed vertices are removed from the problem and lines show mandatory paths. To the right: the result after preprocessing 3.1.2

Note that $S$ and $T$ can be empty or singletons ($t_S = b_S$). Figure 3(g) shows the result after preprocessing the sub-aisle depicted. The vertices of set $S$ are drawn in black whereas the vertices of $T$ are white. Note that the six configurations have either ($t_S b_S$) or ($b_S t_S$) (or both) on one side and either ($t_T b_T$) or ($b_T t_T$) (or both) on the other side, hence the constraints added by the preprocessing.

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

A similar preprocessing was proposed independently by Scholz *et al.* [23], using different constraints to force all the products of one set (S or T) to be picked together.

### *3.1.2. With a distance matrix*

When it is not allowed to add constraints, for example when using Concorde, we can still suppress some vertices. Indeed, the constraints (1) added by the preprocessing are required only if the removal of vertices changes the position of the largest gap in the sub-aisle. Thus, we can still suppress vertices as long as the largest gap remains between $b_T$ and $t_S$ (see Figure 3(h)).

## *3.2. Reducing the number of arcs*

Let's now focus on the arcs. We compute the minimum 1-spanner in terms of number of edges, i.e., we are looking for a subset of the arcs preserving at least one original shortest path between any pairs of required vertices. The rationale is that any solution of the picking problem gives a tour where two products picked successively are linked by a shortest path. So any optimal solution can be found in a 1-spanner of $G$.

**Definition 2.** *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, which can also be seen as a particular case of the Minimum Manhattan Network. 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 $(i, j)$ of required vertices there exists a shortest path which keeps the initial distance $d(i, j)$. Namely we search a graph $H = (V_H, E_H)$ such that: $R \subset V_H$, $V_H \subset V$, $E_H \subset E$ and $\forall i, j \in V_H : \exists$ a $(i, j)$-path $P \in H | \sum_{(uv) \in P} d_{uv} = d(i, j)$.

To compute a minimum 1-spanner, we choose the minimum set of edges with respect to the preceding properties. We use an integer linear program to solve the problem of the **1-spanner**, based on a model for the Minimum Manhattan Network from Benkert *et al.* ([2]). The resulting set of selected edges $E^*$ forms the 1-spanner which can be used to state the Steiner TSP.

The Minimum Manhattan Network is $\mathcal{NP}$-hard [5] but its resolution with MILP turns out to be really fast in practice for the benchmark considered. Finally, note that we don't need to compute the optimal network and any feasible 1-spanner can be used for preprocessing.

## 4. MILP formulation

The Steiner variant of the TSP was proposed by Cornuéjols, Fonlupt and Naddef in 1985 [6]. It was introduced especially to solve problems 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.

We define the Steiner graph from the directed graph modeling an instance (see Figure 1(b)): the required vertices are the products plus the depot and the Steiner vertices are the intersections in the warehouse. We look for a shortest tour in this graph, going through all the required vertices **at least** once.

### *4.1. Flow-based formulation*

We can use the compact single commodity flow formulation proposed by Letchford, Nasiri and Theis [15]. 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 a product.

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 further called **SCFS**, standing for Single Commodity Flow Formulation for a Steiner TSP:

$$(SCFS) \qquad \begin{array}{rlll}
z^* = min & \sum\limits_{(ij)\in A} d_{ij}x_{ij} & & (1) \\
\text{s.t.} & \sum\limits_{j\in\Gamma(i)} x_{ij} \geq 1 & \forall i \in R & (2) \\
& \sum\limits_{j\in\Gamma(i)} x_{ij} = \sum\limits_{j\in\Gamma(i)} x_{ji} & \forall i \in V & (3) \\
& \sum\limits_{j\in\Gamma(i)} y_{ji} - \sum\limits_{j\in\Gamma(i)} y_{ij} = 1 & \forall i \in R \setminus \{0\} & (4) \\
& \sum\limits_{j\in\Gamma(i)} y_{ji} - \sum\limits_{j\in\Gamma(i)} y_{ij} = 0 & \forall i \in V \setminus \{R\} & (5) \\
& y_{ij} \leq n x_{ij} & \forall (ij) \in A & (6) \\
& x_{ij} \in \mathbb{N} & \forall (ij) \in A & (7) \\
& y_{ij} \geq 0 & \forall (ij) \in A & (8)
\end{array}$$

Constraints (2) ensure that each required vertex is visited **at least** once. Constraints (3) 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 (4) impose that the order picker delivers one unit of the commodity to each product while constraints (5) impose that the flow stays the same through a non-required vertex. Constraints (6) link the $y$ to the $x$ variables so that if some flow transits through $(ij)$ then the arc $(ij)$ is chosen.

*Remark* 2. The variables $y$ are real variables (8) but the optimal solution will be integer due to the fact that $y$ represent a flow.

*Remark* 3. We know from Lemma 1 in Letchford, Nasiri and Theis [15] that every optimal solution of the STSP uses an arc at most once and thus satisfies $x_{ij} \leq 1$, $\forall (ij) \in A$. It is therefore sufficient to define $x$ as a positive integer (constraints (7)). So the linear relaxation amounts to $x_{ij} \geq 0$.

This formulation is compact and really sparse, especially thanks to the pre-processings described above. However, we notice that the quality of the lower bound given by the linear relaxation is weaker than the one given by a more standard formulation as it is explained below.

## *4.2. Theoretical study of the formulation*

In this section, we compare the formulation described above with a standard flow-based formulation for the TSP. We consider the problem of finding a shortest tour in the complete and directed graph composed of all the products and where the distance between two vertices is the shortest distance in the warehouse. For sake of simplicity, we use the directed version of the TSP. Thus, we compare the standard single-commodity flow formulation for the TSP (as defined by Gavish & Graves [10]) with the single-commodity flow formulation described above for the Steiner TSP.

We recall the Gavish and Graves formulation (denoted by **SCF**), adapted to a directed graph:

$$(SCF)\quad \begin{array}{lll} z'^{*} = min & \sum\limits_{\substack{i,j\in R\\ i\neq j}} d(i,j)x'_{ij} & \qquad (9)\\ \text{s.t.} & \sum\limits_{\substack{j\in R\\ j\neq i}} x'_{ij} = 1 \qquad \forall i \in R & \qquad (10)\\ & \sum\limits_{\substack{j\in R\\ j\neq i}} x'_{ji} = 1 \qquad \forall i \in R & \qquad (11)\\ & \sum\limits_{\substack{j\in R\\ j\neq i}} y'_{ji} - \sum\limits_{\substack{j\in R\\ j\neq i}} y'_{ij} = 1 \qquad \forall i \in R\setminus\{0\} & \qquad (12)\\ & y'_{ij} \leq n x'_{ij} \qquad \forall i,j\in R,\ i\neq j & \qquad (13)\\ & x'_{ij} \in \{0,1\} \qquad \forall i,j\in R,\ i\neq j & \qquad (14)\\ & y'_{ij} \geq 0 \qquad \forall i,j\in R,\ i\neq j & \qquad (15) \end{array}$$

Constraints (10) and (11) are usual assignment constraints ensuring that each vertex is visited **exactly** once. Constraints (12) ensure that, except for the depot, the salesman deliver one unit at each vertex and retains the rest of the flow. Constraints (13) are *Big-M* constraints linking $y$ and $x$. Finally, the objective (9) is to minimize the total distance of the tour.

To compare both formulations, we define a projection *proj* which projects a fractional solution of (SCF) in the space of (SCFS) by keeping the distance value. The idea is to divide the value $x'$ between two required vertices on all the shortest paths.

**Definition 3** (Projection *proj*)**.**

$$\begin{aligned} proj:\quad \mathbb{R}^{|R|^2} &\rightarrow \mathbb{R}^{|A|}\\ x' &\rightarrow x \end{aligned} \qquad (16)$$

*Where $proj(x')$ is defined by:*

$$x_{uv} = \sum_{i,j\in R} \sum_{\substack{P\in P_{ij}\\ (uv)\in P}} \frac{x'_{ij}}{|P_{ij}|}\ \forall (uv)\in A \qquad (17)$$

By construction, the conservation is checked at each vertex (property (19)) and each required vertex is "visited" at least as many times as in the standard TSP solution (property (18)). A required vertex can be visited more if it lies on a shortest path between two other required vertices.

*Remark* 4 (Properties of *proj*). Let $x' \in \mathbb{R}^{|R|^2}$ and $x = proj(x')$. Then:

$$(i) \quad \sum_{j\in\Gamma(i)} x_{ij} \geq \sum_{j\in R} x'_{ij} \ \forall i \in R \tag{18}$$

$$(ii) \quad \sum_{j\in\Gamma(i)} x_{ij} = \sum_{j\in\Gamma(i)} x_{ji} \ \forall i \in V \tag{19}$$

$$(iii) \quad \sum_{(ij)\in A} x_{ij} d_{ij} = \sum_{i,j\in R} x'_{ij} d(i,j)$$

**Lemma 1.** *The linear relaxation of the Steiner single commodity flow formulation (SCFS) is **weaker** than the linear relaxation of the standard TSP single commodity flow formulation (SCF).*

*Proof.* Every projection of an optimal solution of SCF is feasible in SCFS, so $z'^*_{LP} \geq z^*_{LP}$ and there exists a case for which $z'^*_{LP} \neq z^*_{LP}$ so that SCF is a better formulation than SCFS. The two cases are detailed below:

- $z^*_{LP} \leq z'^*_{LP}$

Let $(x', y')$ an optimal fractional solution of the linear relaxation of (SCF) and $(x, y) = (proj(x'), proj(y'))$. Then $(x, y)$ is a feasible solution for the linear relaxation of (SCFS) since:

- Assignment constraints (2) are respected due to the property (18) of *proj*.
- Conservation constraints are respected due to the property (19) of *proj*.
- For the same reasons, the flow constraints (4) and (5) are respected.
- Constraints (6) is respected: since the transformation is the same on $x'$ and $y'$ to obtain $x$ and $y$, the ratio is kept: $\frac{y_{uv}}{x_{uv}} = \frac{y'_{ij}}{x'_{ij}} \leq n$ so $y_{uv} \leq n x_{uv} \ \forall (uv) \in P_{ij}, \forall i, j \in R$

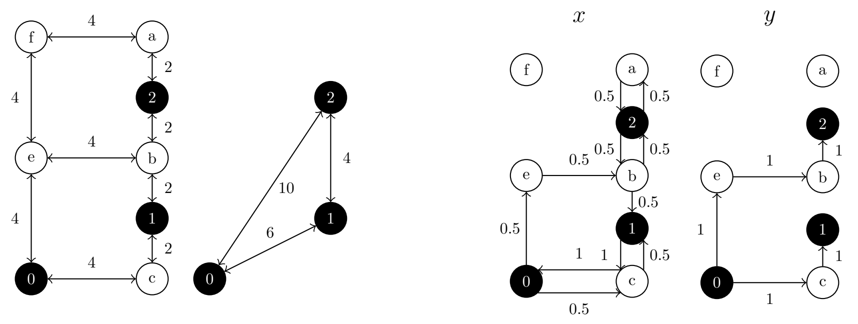
Thus, $(x, y)$ is feasible for $SCFS$ and $z_{LP} = z'_{LP}$.

- $z^*_{LP} \neq z'^*_{LP}$

We consider an example with 3 products and show that $z^*_{LP} \leq 18 < z'^*_{LP} = 20$. Figure 4(a) shows the representations of this example with the Steiner graph and with the complete graph. Solving the linear relaxation of (SCF) leads to $z'^*_{LP} = 20$ because of constraints (10) and (11). On the other side, we can easily build a feasible solution of the linear relaxation of (SCFS) with a cost 18 as shown on Figure 4(b). Thus, in this example we have $z^*_{LP} < z'^*_{LP}$.

□

In the following, we introduce improvements to offset the weakness of the initial formulation and get closer, or overcome, the quality of the Standard TSP single-commodity flow formulation.



(a) On the left: the Steiner graph, on the right: the standard TSP grah

(b) Feasible solution of (SCFS) with $z = 18$

Figure 4: Example with $z^*_{LP} < z'^*_{LP}$

## *4.3. Strengthening of the bound*

Constraints (6) are big-M constraints that make the formulation quite weak and can be improved following Letchford, Nasiri and Theis [15].

Without loss of generality, we can assume that the order picker delivers the unit of commodity due to each required vertex at his first visit. Due to the warehouse structure, a minimum number $n_R(i)$ of required vertices might have to be visited before visiting vertex $i$ (in particular after preprocessing). We can apply a shortest path algorithm to compute $n_R(i)$ and reinforce the bound on $y_{ij}$. Constraints (6) are replaced by:

$$y_{ij} \leq (n - n_R(i))\, x_{ij} \;\forall i \in V, j \in \Gamma(i) \tag{20}$$

## *4.4. Additional cuts*

The Dantzig-Johnson-Fulkerson formulation based on sub-tour elimination constraints is the most well known formulation of the TSP and exhibits a very strong linear relaxation. We consider here a polynomial number of sub-tour elimination constraints depending on the warehouse dimensions rather than the number of products. In the single-flow formulation, the connectivity between all the required vertices is guaranteed. However, because of the big-M constraints (20), the fractional value of $x$ can be really small. Many sets of vertices are violating the sub-tour elimination constraints in practice. 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 $S$. The following valid inequality is added for each cut $(S, \bar{S})$:

$$x\left(S : \bar{S}\right) \geq 1 \tag{21}$$

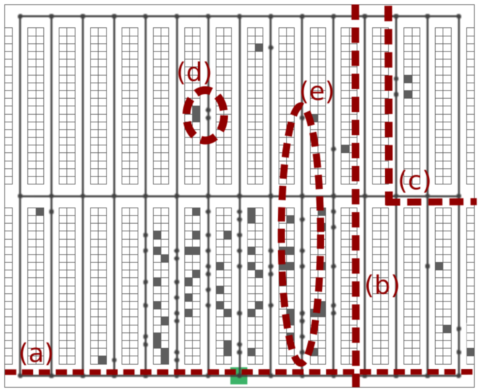


Figure 5: Examples of the different cuts in a warehouse

*Remark* 5. It is easy to see, with constraints of conservation (3) that this inequality is sufficient to impose also $x\left(\bar{S}:S\right)\geq 1$

We introduce the following cuts (see Figure 5):

- Line cuts: horizontal (resp. vertical) cuts $C=(S,\bar{S})$ separating the warehouse horizontally (resp. vertically). See cuts (a) and (b) on Figure 5.
- Corner boxes cuts: we can combine horizontal and vertical cuts to create boxes attached to a corner of a warehouse. See cut (c) on Figure 5.
- Sub-aisle connexity: we define $S\subset V$ as a set of adjacent vertices in the same sub-aisle ($|S|\geq 2$). See cut (d) on Figure 5. There exists at most 6 sets of adjacent vertices in a sub-aisle since there are at most 4 products.
- Cross cuts: S is defined as the interval between the highest product and the lowest product of two adjacent sub-aisles in the same aisle. See cut (e) on Figure 5.

Note that the total number of cuts considered is polynomial in the size of the warehouse (complexity in $\mathcal{O}(hv)$). We can also add valid inequalities implied by the structure of the graph:

- Intersection connexity: a path reaching an intersection (except the depot) can not immediately backtrack in an optimal solution. Namely, if an arc comes in an intersection, i.e, a Steiner vertex, by one side, an arc must go out by **another** side. Thus, we add the following constraints:

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

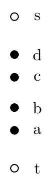
$$x_{ji}\leq\sum_{k\in\Gamma(i)\setminus\{j\}}x_{ik}\quad\forall i\in I,j\in\Gamma(i)$$

- Patterns: we can identify logical implications from the 6 ways to go through a sub-aisle. We recall that a sub-aisle is surrounded by two intersections (denoted $s$ and $t$) and, after the preprocessing, contains at most 4 products (denoted $a$,$b$,$c$,$d$).

  $x_{sd} \Rightarrow x_{dc}$ 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 $x_{bc} \Rightarrow x_{ab} \wedge x_{cd}$



  They are added as linear constraints thanks to the following scheme: $P \Rightarrow Q$ is linearized into $P \leq Q$ and $P \Rightarrow Q \wedge R$ is linearized into $z \geq Q+R-1$, $z \leq Q$, $z \leq R$ and $P \leq z$, where $P$, $Q$, $R$ and $z$ are boolean variables.

Finally, notice that any tour leads to a symmetric one by reversing the direction of traversal. Although constraints can be added to break this symmetry, it did not pay off in our experiments and they were removed.

## 5. Dynamic programming

A dynamic programming algorithm has been proposed by Ratliff and Rosenthal for the picking problem in a rectangular warehouse with two cross-aisles in 1983 [20]. It was extended in 2001 to the case of three cross-aisles by Roodbergen and De Koster [21]. More generally, the rectilinear TSP can be solved by dynamic programming using the very same ideas. An algorithm, proposed by Cambazard and Catusse [4], is proved to have a $O(hn7^h)$ (or more precisely $O(hv7^h)$) runtime complexity where $n$ cities are located on $h$ horizontal lines and $v$ vertical lines. The distance considered between any pair of cities is the $l_1$ (rectilinear or manhattan) distance. This algorithm is directly applicable to the picking problem which can be seen as a specific case. We will now give the key ideas and a summary of this approach in the present section. We refer the reader to [4] for the details and in particular the proofs of correctness and complexity analysis.

We consider the problem in an undirected grid graph such as the one shown in Figure 1(b). The set of vertices located on a vertical aisle or two adjacent vertical aisles is a planar **separator** of this grid graph (e.g. $\{5, 9, 14\}$, $\{d, 9, 14\}$ or $\{d, 10, 14\}$ are separators in the graph of Figure 1(b)). The rationale of the dynamic programming algorithm is that the problem can be split into two subproblems, to the right and to the left of a separator by considering all the possible configurations of the separator (degree parity of the vertices and connected components described below). The algorithm builds a **tour subgraph** that can be directed as a post-processing step to obtain a picking tour. An example of such tour subgraph is shown on Figure 6.

### *5.1. States*

A state of the dynamic program is a possible configuration of a separator. In the following, such a state $\omega$ is denoted $\omega = \{(x_1, \ldots, x_h), (c_1, \ldots, c_h)\}$ where $x_i \in \{U, E, 0\}$ and $c_i$ are respectively the parity label and the connected component of the $i$-th vertex of $\omega$. We use the same notation as Ratliff and Rosenthal [20] to describe degree parities: even = E, odd = U (uneven) and zero = 0. Connected components are described by their indices or "$-$" for a zero degree. Figure 6 gives an example of a state where all vertices have an even degree and belong to two distinct connected components.

### *5.2. Transitions*

There are two types of transitions between states: vertical and horizontal transitions, corresponding to the decisions made on vertical or horizontal edges of the grid graph. Horizontal transitions are of three kinds: no edge, a single edge or a double edge. Vertical transitions are of the six kinds identified by Ratliff and Rosenthal [20]: no edge, a single edge, a single double edge, two double edges connected to the top vertex, two double edges connected to the bottom vertex and four double edges defined by the largest gap (see Figure 3).

### *5.3. Outline of the algorithm*

Algorithm 1 processes the edges of the grid graph from bottom to top and then from left to right (line 3). Typically on Figure 1(b), the edges would be considered in the following order: (4,8), (8,13), (4,5), (8,9), (13,14), (5,9) and so on. All the states obtained after adding $l$ transitions (denoted $Layer_l$) belong to the l-th layer and the algorithm can be seen as a shortest path algorithm in a layered graph (see Figure 7). From any state (line 5), the possible transitions are considered (line 6); three or six depending if the edge considered is a vertical or horizontal one. We denote by $T(\omega, l)$ the value of the shortest path to reach state $\omega$ located on layer $l$. Lines 7-14 update the possible states of the next layer (l+1) by extending the considered state $\omega$ of layer $l$ with the considered transition $tr$. The new state $\omega'$ might not be a valid tour subgraph and it is checked line 8. For instance, a partial tour subgraph is not valid if a vertex is

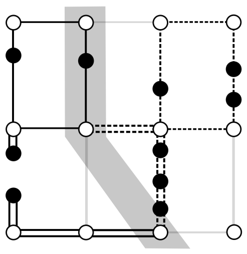


Figure 6: The black edges represent a partial tour subgraph and the dashed edges represent one possible completion to a complete tour subgraph. The white vertices in the gray area represents the current state {(E,E,E)(1,1,2)}.

```
1: $\omega_0 \leftarrow \{(0,\dots,0),(-,\dots,-)\}$; $T(w_0,0)=0$; $Layer_0 \leftarrow \{w_0\}$
2: $l \leftarrow 0$
3: for each edge $e$ of the grid graph from bottom to top and left to right do
4:    $Layer_{l+1} \leftarrow \emptyset$
5:    for each state $\omega \in Layer_l$ do
6:       for each possible transitions $tr$ for $e$ do
7:          $\omega' \leftarrow \omega + tr$
8:          if check$(\omega', l+1)$ then
9:             if $\omega' \in Layer_{l+1}$ then
10:               if $T(\omega', l+1) > T(\omega, l) + length(tr)$ then
11:                  $T(\omega', l+1) \leftarrow T(\omega, l) + length(tr)$
12:            else
13:               $T(\omega', l+1) \leftarrow T(\omega, l) + length(tr)$
14:               $Layer_{l+1} \leftarrow Layer_{l+1} \cup \{\omega'\}$
15:   $l \leftarrow l+1$
16: $w_{opt} \leftarrow \text{argmin}_{\omega \in L_{hv}} T(\omega, hv)$
17: return $w_{opt}$
```

**Algorithm 1:** Dynamic Programming algorithm for rectangular picking

left with an odd number of incident edges, if a product is not collected (zero degree) or if it has more than one connected component on the last layer (since the final tour subgraph must be connected). A shortest partial tour subgraph might be already known to reach $\omega'$ and this is checked line 10.

An illustrative execution of the algorithm is given Figure 7, where a particular path is outlined showing the relation between states and partial tour subgraphs.

## 6. Experimental results

### 6.1. Implementation

We used the CPLEX Java API (version 12.6) to solve the different linear programs. An initial upper bound is given as "warm start" and obtained with the Lin-Kernighan heuristic [13] implemented in the software LKH (freely available at `http://webhotel4.ruc.dk/~keld/research/LKH/`). This gives a better comparison with the TSP solver Concorde that takes advantage of an initial accurate upper bound[1]. The instances solved come from an academic benchmark proposed by Theys, Dullaert and Herroelen [25]. Some of these instances seem bigger than realistic data, but allow to test the limits of the algorithms and to perform a comparison with previously published results on the exact same instances [24]. Experiments were performed 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.

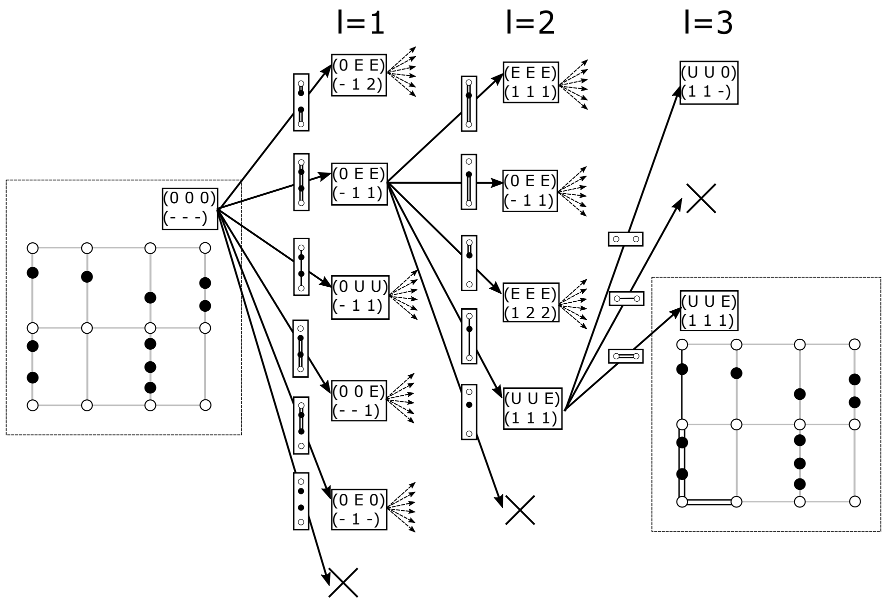


Figure 7: Example of the graph underlying the dynamic programming algorithm. Each layer is identified with a value of l. Three transitions are possible from each state. The partial toursubgraph obtained by following the black path is shown on the bottom right corner.

| parameter | values |
|---|---|
| the number of vertical aisles | { 5, 15, 60 } |
| the number of cross-aisles | { 3, 6, 11 } |
| the number of products in the picking list | { 15, 60, 240 } |
| the storage policy | { volume (V), random (R) } |
| the location of the depot in the warehouse | { central, decentral } |

Table 1: Parameters and values of the instances

## *6.2. Description of the instances*

The benchmark of Theys *et al.* contains 108 classes of instances, where a class is defined by 5 parameters described in Table 1. The storage policy is either random-based or volume-based. A random policy means the products are randomly affected in the warehouse. A volume-based policy means a twenty percent of the most demanded items are located near the first cross-aisle. For more details on the instance, see Theys *et al.* [24].

Early experiments showed that the location of the depot (central or decentral) does not affect the efficiency of our models. Thus, we report our results on the 54 classes with a central depot where each class contains 10 instances.

## *6.3. Results*

The performances of the following algorithms are compared:

|  | Total | Storage policy |  | # aisles |  |  | # cross-aisles |  |  | # products |  |  |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
|  |  | R | V | 5 | 15 | 60 | 3 | 6 | 11 | 15 | 60 | 240 |
| TSP | 20580 | 20580 | 20580 | 20580 | 20580 | 20580 | 20580 | 20580 | 20580 | 240 | 3660 | 57840 |
| TSP+ | 6676 | 9228 | 4123 | 2319 | 6042 | 11665 | 3744 | 7238 | 9046 | 217 | 2350 | 17460 |
| Evolution | -68% | -55% | -80% | -89% | -71% | -43% | -82% | -65% | -56% | -9% | -36% | -70% |
| Steiner | 768 | 790 | 746 | 192 | 481 | 1633 | 355 | 705 | 1245 | 678 | 743 | 884 |
| Steiner+ | 474 | 543 | 405 | 157 | 347 | 917 | 267 | 456 | 698 | 216 | 446 | 760 |
| Evolution | -38% | -31% | -46% | -18% | -28% | -44% | -25% | -35% | -44% | -68% | -40% | -14% |

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

|  | Total | Storage policy |  | # aisles |  |  | # cross-aisles |  |  | # products |  |  |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
|  |  | R | V | 5 | 15 | 60 | 3 | 6 | 11 | 15 | 60 | 240 |
| SCFS | 46.4% | 43.12% | 49.68% | 34.85% | 45.8% | 58.53% | 41.95% | 47.13% | 50.11% | 46.9% | 45.24% | 47.05% |
| SCFS_PP | 39.32% | 37.41% | 41.22% | 24.49% | 38.91% | 54.55% | 33.23% | 40.68% | 44.03% | 44.74% | 40.55% | 32.65% |
| SCFS+ | 1.4% | 1.68% | 1.13% | 2.7% | 1.05% | 0.46% | 1.72% | 1.28% | 1.21% | 0.4% | 1.21% | 2.6% |

Table 3: Average gap of linear relaxations for single-commodity flow formulations

- SCFS: the basic Steiner single commodity flow formulation
- SCFS_PP: SCFS with preprocessing of section 3
- SCFS+: SCFS with preprocessing and the additional valid inequalities
- SCF+: the standard single commodity flow formulation with vertex preprocessing 3.1.1
- CDE: Concorde
- CDE+: Concorde with preprocessed input (described in section 3.1.2)
- PDYN: dynamic programming (section 5)

To start with, we report an analysis of the problem's size, which is the biggest advantage of the SCFS formulation. Then, we look at the strength of the linear relaxation of each formulation and note that SCFS strongly benefits from the improvements proposed. In the end, we compare resolution times. The tables report the average value of the quantity studied (size, gap, cpu times) for all instances restricted to the value of the parameter given as the column header.

*Size analysis.* Table 2 shows the number of arcs in the TSP case (complete graph) and in the Steiner graph with and without the preprocessing 3. The lines "Evolution" show the percentage of arcs removed when the preprocessing is applied. We notice that the preprocessing is really more efficient in the TSP case due to the completeness of the graph. We also notice that the number of aisles, cross-aisles and products have an opposite effect depending on the solver.

Finally note that the number of arcs in the Steiner graph is much smaller than in the complete graph which significantly improve the memory scaling of the MILP formulations.

|  | Total | Storage policy |  | # aisles |  |  | # cross-aisles |  |  | # products |  |  |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
|  |  | R | V | 5 | 15 | 60 | 3 | 6 | 11 | 15 | 60 | 240 |
| SCFS+ | 0.65% | 1.01% | 0.29% | 1.33% | 0.36% | 0.27% | 0.81% | 0.53% | 0.61% | 0.04% | 0.34% | 1.59% |
| SCF+ | 6.44% | 7.47% | 5.48% | 2.39% | 4.73% | 13.3% | 7.6% | 6.27% | 5.34% | 0.37% | 5.09% | 15.36% |
| CDE+ | $< 0.1\%$ |  |  |  |  |  |  |  |  |  |  |  |

Table 4: Average gap of root node relaxations for single-commodity flow formulations and Concorde

|  | Total | Storage policy |  | # aisles |  |  | # cross-aisles |  |  | # products |  |  |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
|  |  | R | V | 5 | 15 | 60 | 3 | 6 | 11 | 15 | 60 | 240 |
| SCFS+ | 18 | 18 | 0 | 1 | 4 | 13 | 1 | 2 | 15 | 0 | 0 | 18 |
| SCF+ | 136 | 88 | 48 | 19 | 34 | 83 | 51 | 41 | 44 | 0 | 26 | 110 |
| PDYN | 180 | 90 | 90 | 60 | 60 | 60 | 0 | 0 | 180 | 60 | 60 | 60 |
| # instances | 540 | 270 | 270 | 180 | 180 | 180 | 180 | 180 | 180 | 180 | 180 | 180 |

Table 5: Number of unsolved instances after 30 minutes

*Analysis of lower bounds.* Table 3 compares the average gap between the **linear relaxation** and the optimal value for each parameter and for the different single-commodity flow formulations. The gap is computed as $\frac{z^*-z^*_{LP}}{z^*} \times 100$. The results clearly demonstrate that the improvements proposed are very effective to strengthen the linear relaxation. The gap moves from 46% to 1% in average without increasing significantly the computing time which moves from 0.07 second to 0.2 second in average. To compare our results to Concorde (see Table 4), we observe the gap between the **lower bound at the root node** of the search tree and the optimal value. This lower bound is better than the linear relaxation since CPLEX and Concorde apply many techniques to improve it.

SCFS+ has a strong linear relaxation despite the fact that the initial formulation SCFS is weaker than all the other ones. In practice, we observe that the linear relaxation of the improved Steiner single commodity flow formulation (SCFS+) is significantly **stronger** than the linear relaxation of the improved standard single commodity flow formulation (SCF+).

*Performances.* We set a time-limit of 30 minutes for SCFS+ and SCF+. Some instances were unsolved in this time limit. Table 5 shows the number of unsolved instances after 30 minutes of processing depending on the different parameters.

The improved Steiner formulation completely outperforms the standard compact TSP formulation. It still cannot solve some instances but they are from only 5 classes from the biggest ones and only with a random policy (15_11_240_R, 5_11_240_R 60_11_240_R, 60_3_240_R, 60_6_240_R). On the other hand, instances from 16 classes cannot be solved by the SCF solver. The only param-

|  | Total | Storage policy |  | # aisles |  |  | # cross-aisles |  |  | # products |  |  |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
|  |  | R | V | 5 | 15 | 60 | 3 | 6 | 11 | 15 | 60 | 240 |
| SCFS+ | 1.3% | 1.3% | - | 0.85% | 0.46% | 1.59% | 0.23% | 0.29% | 1.51% | - | - | 1.3% |
| SCF+ | 19.63% | 19.1% | 20.51% | 10.19% | 18.8% | 23.45% | 16.37% | 21.19% | 23.91% | - | 7.79% | 23.47% |

Table 6: Average gap between best upper and lower bounds for unsolved instances

|  | Total | Storage policy |  | # aisles |  |  | # cross-aisles |  |  | # products |  |  |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
|  |  | R | V | 5 | 15 | 60 | 3 | 6 | 11 | 15 | 60 | 240 |
| SCFS+ | 36.07 | 61.21 | 12.62 | 23.89 | 56.96 | 27.12 | 3.44 | 35.88 | 71.69 | 0.07 | 4.05 | 111.64 |
| PDYN | 0.27 | 0.28 | 0.27 | 0.05 | 0.16 | 0.61 | 0 | 0.54 | - | 0.24 | 0.27 | 0.30 |
| CDE | 6.86 | 12.8 | 0.93 | 17.61 | 2.11 | 0.88 | 14.82 | 3.89 | 1.88 | 0.01 | 0.13 | 20.45 |
| CDE+ | 1.60 | 2.97 | 0.23 | 3.45 | 1.04 | 0.30 | 2.20 | 1.55 | 1.04 | <0.01 | 0.1 | 4.68 |

Table 7: Average time of optimal resolution (in seconds) for instances solved in less than 30 minutes with each solver

eter guaranteeing an optimal resolution is # products = 15. Moreover, as the Table 6 shows, the gap between the lower and upper bounds is really smaller for the SCFS+, which indicates that the upper bound can be used as a good feasible solution.
The dynamic program solves all the instances with less than 11 cross-aisles. This is a known limitation of this approach since the algorithm has an exponential complexity in the number of cross-aisles.

To compare resolution time, we choose the instances solved in less than 30 minutes by solver SCFS+. Formulation SCF+ has too many unsolved instances after this time limit so it appears irrelevant to include it in the comparison. Table 7 shows the numerical results. For each parameter, SCFS+ is slower than Concorde and dynamic programming. However, on many instances the computing time is reasonable.
We also included results for solver CDE to show the impact of the preprocessing on the Concorde solver. Note that, with the preprocessing, any instance of the entire benchmark of [24] can be solved optimally in less than 1 minute by Concorde.
The dynamic programming approach is the quickest since all the accepted instances are solved in less than a second.

*Instances from Scholz et al.* Scholz *et al.* introduced other instances for the case of a single-block layout [23].

| parameter | values |
|---|---|
| number of products in a sub-aisle | 45 |
| number of cross-aisles | 2 |
| number of vertical aisles | { 5, 10, 15, 20, 25, 30 } |
| number of products in the picking list | { 30, 45, 60, 75, 90 } |
| location of the depot in the warehouse | decentral |

Table 8: Parameters and values of the Scholz instances

The parameters are given in table 8 and form 30 classes of instances. For each one of their classes, we generated 10 instances of the same size. As expected, since these instances have only one block, the dynamic program is extremely efficient and solves any instance optimally in less than a second. Our MILP model also provides optimal solution for all instances almost instantaneously. Indeed, the maximal time of resolution is 6.7 seconds for the biggest instance

|  | Total | # aisles |  |  |  |  |  | # products |  |  |  |  |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
|  |  | 5 | 10 | 15 | 20 | 25 | 30 | 30 | 45 | 60 | 75 | 90 |
| Scholz | 167.7 | 0.11 | 1.21 | 10.54 | 170.91 | 314.56 | 508.87 | 45.53 | 87.56 | 149.03 | 233.24 | 323.14 |
| SCFS+ | 0.28 | 0.06 | 0.09 | 0.13 | 0.34 | 0.54 | 0.54 | 0.07 | 0.13 | 0.25 | 0.38 | 0.6 |
| PDYN | < 0.001 | < 0.001 |  |  |  |  |  |  |  |  |  |  |

Table 9: Average time of optimal resolution (in seconds) for "Scholz instances", solved in less than 30 minutes with each solver

(30 aisles and 90 products) and the average on all the instances is 0.28 seconds. Moreover, the model introduced by Scholz *et al.* fails to find the optimal solution for 44 instances. Table 9 details the computation time to find the optimal solution for these solvers. Please note that our instances are not exactly the same than those solved by Scholz *et al.* but were generated with the same parameters.

*Discussion.* A number of fast heuristics have been proposed in the past to solve the order picking problem without any guarantee of optimality. And some of these heuristics have been improved to achieve better solution at the expanse of their running time[24]. Note that the dynamic program proposed in this paper can solve instances up to 6 cross-aisles with an average time of 0.27 seconds *i.e.*, the same order of running time reported in Theys et al (between 0.34 and 0.43 seconds for 3 and 6 cross-aisles) for advanced heuristics. We also showed that the use of Concorde with the preprocessing we propose in this paper gives excellent results even beyond 6 cross-aisles. In average, the computation time is less than 2 seconds to compute an exact solution and this method can be applied on instances of any size. The use of heuristics is therefore not relevant in our opinion if we are only concerned with minimizing the distance of the tour, since the dynamic program or Concorde with preprocessing provide an exact solution in a really small amount of time. However, these methods share a weakness with the dedicated heuristics: they cannot easily accommodate side-constraints. Our MILP model is not as efficient as the other methods but it outperforms previously proposed MILP and solve many instances in a reasonable amount of time. Thus, it is relevant to employ it when the user wants an exact solution and has specific requirements that can be modeled with linear constraints.

## 7. Conclusion

We have studied two exact algorithms for the picking problem based on dynamic programming and mixed integer linear programming.
The first approach was previously proposed for warehouses with up to three cross-aisles. We extend it to any number of cross-aisles which has often been mentioned but never done before. This algorithm proves to be extremely efficient for realistic size of warehouses. However, it can not accommodate side constraints such as precedences, flow directions or multiple depots: a MILP is better suited to deal with these requirements.

With this in mind, 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 has a weak linear relaxation. As a result, it is also inefficient in practice.
Scholz *et al.* proposed a new formulation, with improvement by preprocessing. However, their model is rather complex and does not yet provide convincing results regarding its efficiency when the layout grows[23]. We thus propose a number of improvements by taking advantage of the warehouse structure. These improvements are based on valid inequalities and procedures to significantly reduce the instance's size without loosing optimality. The resulting model remains sparse and exhibits a strong linear relaxation in practice. It outperforms the compact TSP model and solves very large instances almost as efficiently as 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 [24] can thus be solved to optimality very efficiently without the need of the heuristics proposed by the same authors.

The analysis of the results showed that the improvement was stronger when the products were stored with a volume policy. It may be a promising track to solve the picking problem jointly with other warehouse issues such as storage policy or batching. Valle *et al.* follow this track in a recent paper [27] as did Won and Olafsson a few years ago[29].

## References

[1] David Applegate, William Cook, and André Rohe, *Chained lin-kernighan for large traveling salesman problems*, INFORMS Journal on Computing **15** (2003), no. 1, 82–92.

[2] Marc Benkert, Alexander Wolff, Florian Widmann, and Takeshi Shirabe, *The minimum manhattan network problem: approximations and exact solutions*, Computational Geometry **35** (2006), no. 3, 188–208.

[3] 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** (1998), no. 3, 496–546.

[4] Hadrien Cambazard and Nicolas Catusse, *Fixed-parameter algorithms for rectilinear steiner tree and rectilinear traveling salesman problem in the plane*, European Journal of Operational Research (2018).

[5] Francis YL Chin, Zeyu Guo, and He Sun, *Minimum manhattan network is* NP*-complete*, Discrete & Computational Geometry **45** (2011), no. 4, 701–722.

[6] Gérard Cornuéjols, Jean Fonlupt, and Denis Naddef, *The traveling salesman problem on a graph and some related integer polyhedra*, Mathematical programming **33** (1985), no. 1, 1–27.

[7] George Dantzig, Ray Fulkerson, and Selmer Johnson, *Solution of a large-scale traveling-salesman problem*, Journal of the operations research society of America **2** (1954), no. 4, 393–410.

[8] René De Koster and Edo Van der Poort, *Routing orderpickers in a warehouse: a comparison between optimal and heuristic solutions*, IIE transactions **30** (1998), no. 5, 469–480.

[9] Bernhard Fleischmann, *A cutting plane procedure for the travelling salesman problem on road networks*, European Journal of Operational Research **21** (1985), no. 3, 307–317.

[10] Bezalel Gavish and Stephen C Graves, *The travelling salesman problem and related problems* (1978).

[11] Michael Hahsler and Kurt Hornik, *Tsp infrastructure for the traveling salesperson problem*, Journal of Statistical Software **23** (2007), no. 1, 1–21.

[12] Randolph W Hall, *Distance approximations for routing manual pickers in a warehouse*, IIE transactions **25** (1993), no. 4, 76–87.

[13] Keld Helsgaun, *An effective implementation of the lin–kernighan traveling salesman heuristic*, European Journal of Operational Research **126** (2000), no. 1, 106–130.

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

[15] 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** (2013), no. 1, 83–92.

[16] 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** (2003), no. 5, 827–832.

[17] CS Orloff, *A fundamental problem in vehicle routing*, Networks **4** (1974), no. 1, 35–64.

[18] 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** (2006), 93–106.

[19] Charles G Petersen, *An evaluation of order picking routeing policies*, International Journal of Operations & Production Management **17** (1997), no. 11, 1098–1111.

[20] H Donald Ratliff and Arnon S Rosenthal, *Order-picking in a rectangular warehouse: a solvable case of the traveling salesman problem*, Operations Research **31** (1983), no. 3, 507–521.

[21] Kees Jan Roodbergen and René De Koster, *Routing order pickers in a warehouse with a middle aisle*, European Journal of Operational Research **133** (2001), no. 1, 32–43.

[22] Kees Jan Roodbergen and René De Koster, *Routing methods for warehouses with multiple cross aisles*, International Journal of Production Research **39** (2001), no. 9, 1865–1883.

[23] André Scholz, Sebastian Henn, Meike Stuhlmann, and Gerhard Wäscher, *A new mathematical programming formulation for the single-picker routing problem*, European Journal of Operational Research **253** (2016), no. 1, 68–84.

[24] 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** (2010), no. 3, 755–763.

[25] Christophe Theys, Wout Dullaert, and Willy Herroelen, *Routing order pickers in multiple block warehouses*, Proceedings of the bivec-gibet transport research day, 2007/hilferink, pieter [edit.], 2007, pp. 10–30.

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

[27] Cristiano Arbex Valle, John E Beasley, and Alexandre Salles da Cunha, *Modelling and solving the joint order batching and picker routing problem in inventories*, International symposium on combinatorial optimization, 2016, pp. 81–97.

[28] TS Vaughan, *The effect of warehouse cross aisles on order picking efficiency*, International Journal of Production Research **37** (1999), no. 4, 881–897.

[29] J Won and S Olafsson*, *Joint order batching and order picking in warehouse operations*, International Journal of Production Research **43** (2005), no. 7, 1427–1442.