RAIRO-Oper. Res. 45 (2011) 179–207
DOI: 10.1051/ro/2011110

RAIRO Operations Research
www.rairo-ro.org

TREE BASED MODELS AND ALGORITHMS FOR THE PREEMPTIVE ASYMMETRIC STACKER CRANE PROBLEM

Hervé Kerivin[1], Mathieu Lacroix[2], Alain Quilliot[2] and Hélène Toussaint[2]

Abstract. In this paper we deal with the preemptive asymmetric stacker crane problem in a heuristic way. We first present some theoretical results which allow us to turn this problem into a specific tree design problem. We next derive from this new representation an integer linear programming model together with simple and efficient greedy and local search heuristics. We conclude by presenting experimental results which aim at both testing the efficiency of our heuristic and evaluating the impact of the preemption hypothesis.

Keywords. Preemptive stacker crane problem, routing, local search, heuristics.

Mathematics Subject Classification. 99-XX.

1. Introduction

Pickup and delivery problems, which consist in scheduling the transportation of sets of goods and/or passengers from origin nodes to destination nodes while using a given set of vehicles, have been intensively studied for decades. Many variants have been considered and one can refer to [7,29,33] for surveys on these problems and methods. Among all the pickup and delivery like problems which have been addressed by researchers, the *stacker crane* problem is characterized by the fact that only one vehicle is involved, which can deal with only one demand unit at the same time.

Received November 11, 2009. Accepted June 16, 2011.

[1] Department of Mathematical Sciences, Clemson University, CLEMSON, O-326, Martin Hall, Clemson, 29634 SC, USA

[2] LIMOS, CNRS UMR 6158, Université Blaise-Pascal, Clermont-Ferrand II, Complexe Scientifique des Cézeaux, 63177 Aubière Cedex, France. alain.quilliot@isima.fr

Article published by EDP Sciences

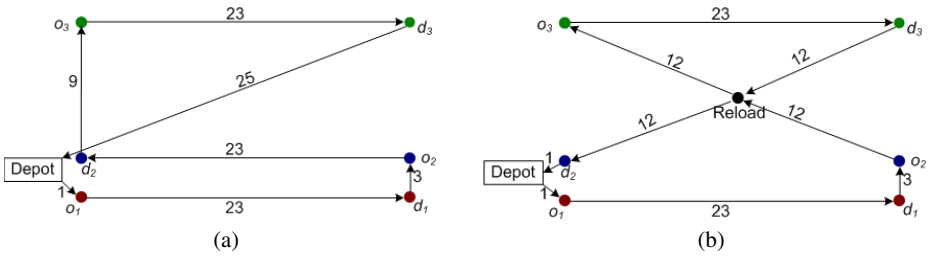


FIGURE 1. (a) Solution without reload; (b) solution with one reload.

An informal description of the *stacker crane problem* (**SCP**) comes as follows: given some transit network G whose oriented links or arcs are endowed with lengths or costs which to any arc (x, y), makes correspond some cost D_{xy}, and which is provided with some specific *Depot* node; we are required to schedule the route of a single vehicle V, which is required to address a *Demand* set K, each demand $k \in K$ being defined by some origin node o_k and by some destination node d_k. Namely, addressing the demand k means transporting some unique load unit from o_k to d_k, the capacity of V being such that V cannot contain more than one load unit at a given time. Thus, scheduling V means designing a tour Γ inside the network G which is going to start and end in *Depot* while making possible for V to handle every demand $k \in K$. Solving the *stacker crane problem* will mean computing this tour in such a way that it is the shortest possible. In the *Preemptive Stacker Crane Problem* (**PSCP**), any load unit may be dropped (unloaded) at any node x of the transit network G, before being later reloaded. This unload/reload process may be performed several times before the load unit reaches its destination node. In case the costs $D_{xy}(x, y \in X)$ are symmetric, we talk about Symmetric **SCP** and conversely if no symmetry hypothesis is made we talk about Asymmetric **SCP.**

Introducing the preemption hypothesis allows us in many cases to reduce the optimal tour cost (length). Figure 1a shows a non preemptive tour which addresses three demands 1, 2 and 3 according to this order. Its cost is 107 (distances used for this example are Euclidean distances). Figure 1b shows a tour which addresses the same demands while dealing with demand 2 in a preemptive way: the vehicle unloads the load unit of demand 2 at the node *Reload* that allows it to deal with demand 3. Finally it goes back to node *Reload* to complete its tour while bringing the load unit of demand 2 to its destination. This tour cost is 99, so we obtain some 7.5% of improvement with one reload only

The *stacker crane* problem was first introduced by Frederickson *et al.* in [18], under its non preemptive symmetric form. These authors proved its NP-hardness by using a reduction from the TSP [24,25]. They also got a 9/5-approximation scheme for this problem. Atallah and Kosaraju [5] were the first to consider the preemptive version of the symmetric **SCP**. They studied both non-preemptive and preemptive versions of the symmetric **SCP** when the underlying graph is an elementary path or an elementary cycle. They proved that in such a case,

both versions are polynomial-time solvable. Frederickson and Guan [16,17] studied both preemptive and non-preemptive versions of the symmetric **SCP** when the underlying graph is a tree Lacroix [23] gives a polyhedral analysis of the preemptive **SCP.** Several variants of the pickup and delivery problems closely related to the **SCP** have been studied. We mention the *pickup and delivery traveling salesman problem* (PDTSP) which corresponds to the non-preemptive stacker crane problem where no capacity constraint is taken into account: see Renaud *et al.* [32,33] Rodin and Ruland [34] and also [13,22] The asymmetric version of the PDTSP was handled through polyhedral approaches and branch-and-cut algorithms [3,4,6,20] as well as through heuristics [10,19,27]. Hernández-Pérez and Salazar-González [21] considered the PDTSP with capacity constraints. Also, authors have considered additional constraints such as time windows [29], precedence constraints imposed to demand processes [14,15] or LIFO loading policy [11], and also a stacker crane problem extension involving several origins and destinations [30]. This extension, named the *swapping problem* [1,2,8,9], belongs to the class of *many-to-many* pickup and delivery problems (see [7]). Finally, we mention some models which involve *transshipments*: pickup and delivery problem with transfers (PDPT) [11,12], with time-windows and transshipments (PDPTWT) [26,31] and with reloads (RPDP) [28].

The focus of this paper will be on the *asymmetric preemptive stacker crane* problem, which we shall denote by **APSCP**. We are first going (Sect. 2) to set our problem in a formal way. Next (Sect. 3) we shall prove some structural results which will allow us to turn the problem into an unconstrained tree design problem. This reformulation of the problem will lead us to cast it into a specific integer linear programming model, and to design (Sect. 4) in a natural way a local search heuristic scheme whose implementation and tests are discussed in Section 5 providing us with satisfactory numerical results.

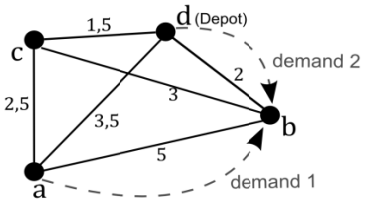
2. A formal description of the APSC problem

This section aims at providing a formal model of the APSCP This description will help us in Section 3 in making clear that the search for an optimal solution of APSCP may be restricted to a specific subset of feasible solutions which admit what we shall call a *tree representation*

2.1. Modelling the asymmetric preemptive stacker crane problem (APSCP)

As told in Section 1, the *asymmetric preemptive stacker crane problem* can be described as follows:

- a single vehicle V is required to address a set K of transportation demands, while performing some tour inside a given oriented transit network G. Any demand $k \in K$ is expressed as a pair (o_k, d_k) of nodes of G, according to the following semantics:
  - o_k is the origin node of k;



		X_O		X_D		X_R			
	Depot	o_1	o_2	d_1	d_2	*a*	*b*	*c*	*d*
Depot	0	3.5	0	2	2	3.5	2	1.5	0
o_1	3.5	0	3.5	5	5	0	5	2.5	3.5
o_2	0	3.5	0	2	2	3.5	2	1.5	0
d_1	2	5	2	0	0	5	0	3	2
d_2	2	5	2	0	0	5	0	3	2
a	3.5	0	3.5	5	5	0	5	2.5	3.5
b	2	5	2	0	0	5	0	3	2
c	1.5	2.5	1.5	3	3	2.5	3	0	1.5
d	0	3.5	0	2	2	3.5	2	1.5	0

FIGURE 2. Deriving a logical node set X from a physical network G.

  - d_k is the destination node of k;
  - V must transport exactly one load unit from o_k to d_k;
- the load capacity of V is equal to 1;
- V is allowed to address a demand k in K in a preemptive way: it may, while carrying its load L stop at some node x, unload L, deal with other demands and next come back to x, load again L and keep on with the handling of L. Such an intermediate node is then called a *reload node* for the demand k;
- V starts and ends its tour in a "*Depot*" node, and try to do it as fast as possible, in the sense of a cost (length) function which is supposed to be defined on the arcs of the network G.

In order to get a formal model of this problem, we make copies of the original *physical* nodes of the network G in such a way that the nodes *Depot*, $o_k d_k$, $k \in K$, and the possible reload nodes **become all distinct**. That means that we deal with a *logical* node set X which may be written according to some **partition**: $X = \{Depot\} \cup X_O \cup X_D \cup X_R$, in such a way that:

- $X_O = \{o_k,\ k \in K\}$;
- $X_D = \{d_k,\ k \in K\}$;
- X_R contains a copy of every element in $\{Depot\} \cup X_O \cup X_D$ together with a set of other possible reload nodes.

Then the original cost function which was defined on the arc set of the network G gives rise, through a shortest path computing process, to a $X \cdot X$ indexed (*shortest path*) *distance* matrix D, such that if $x \in \{Depot\} \cup X_O \cup X_D$ and if x' is the copy of x in X_R, then $D_{xx'} = 0$. Doing this allows us to get rid of the original graph structure: from now on, we deal with a node set X, with a distance matrix D, and we suppose that D satisfies the **triangle inequality property**, but that it does not need to be symmetric.

The left part of Figure 2 represents a network G with 4 *physical* nodes a, b, c and d, and 6 symmetrical links, given together with 2 demands $\mathbf{1}$ = (a, b) and $\mathbf{2} = (d, b)$. The right part represents the resulting $X \cdot X$ matrix D.

2.2. Formal description of a APSCP solution

We start defining feasible solutions as sequences of objects which we call *labelled links.* In order to do it we first need to introduce some notations about sequences.

2.2.1. *Preliminary notations about sequence*

Let $\Gamma = \{x_1, \ldots, x_n\}$ be a sequence of objects x_i, $i = 1 \ldots n$:

- we denote by Succ(Γ, x_i) (Pred(Γ, x_i)), the successor (predecessor) x_{i+1} (x_{i-1}) of x_i in Γ, and by Rank(Γ, x_i) the rank of x_i in Γ. A sequence with only one element x is denoted by $\{x\}$ and the empty sequence is denoted by *Nil* The number n of elements of Γ is denoted by $|\Gamma|$;
- we call *subsequence* of Γ any sequence $\Gamma' = \{x_{i_1}, \ldots, x_{i_p}\}$ with $i_1 < i_2 < \ldots < i_p$ and $1 \leqslant i_p \leqslant n$; (remark: it comes from this definition that two consecutive elements of Γ' may not be consecutive in Γ);
- if x_i and x_j are two elements of Γ such that $i \leqslant j$, then we denote by $I(\Gamma, x_i, x_j)$ the subsequence (or *segment*) $\{x_i, \ldots, x_j\}$ of Γ which is defined by all z such that $i \leqslant \text{Rank}(\Gamma, z) \leqslant j$;
- the first (last) element of Γ is denoted by First(Γ) (Last(Γ));
- we denote by $\oplus$ the *concatenation* operator, which takes two sequences $\Gamma = \{x_1, \ldots, x_n\}$ and $\Gamma' = \{y_1, \ldots, y_m\}$ and concatenates them into a unique sequence $\Gamma \oplus \Gamma\,' = \{x_1, \ldots, x_n, y_1, \ldots, y_m\}$;
- we call *cut* of Γ any decomposition of Γ as a concatenation $\Gamma' \oplus \Gamma''$.

2.2.2. *Labelled links, tours and valid tours*

We represent a tour of a vehicle V using *labelled links:* a labelled link is a triple $r = (x, y, k)$, where x and y are nodes of X and k is a label in the set $\{0\} \cup K$: $x(y)$ is called the *starting node* (*ending node*) of the labelled link r and is denoted by Start(r) (End(r)); k is called the *label* of r and is denoted by Label(r). So the labelled link (x, y, k) is going to represent a move of the vehicle V from the node x to the node y, V being empty if $k = 0$ or loaded with the unit load of demand k otherwise.

A *tour* defined on X is a sequence Γ of labelled links. For such a tour Γ, and for any label k in $\{0\} \cup K$, we denote by Γ^k the labelled link sequence which derives in a natural way from Γ by only considering the labelled links r such that Label(r) = k, and we call it the *deriving subsequence* of Γ related to the label k. The cost of Γ is then defined in a natural way as the quantity:

$$\text{Cost}\,(\Gamma) = \sum_{r \in \Gamma} D_{\text{Start}(r),\text{End}(r)}$$

Clearly, not any tour Γ is likely to express the activity of a vehicle V which would conveniently handle every demand $k \in K$. In order to get such a property, we need to impose Γ to be *valid*, which will mean that:

- for any consecutive pair of labelled links r, $r' = \text{Succ}(\Gamma, r)$ in Γ, we have End(r) = Start(r');

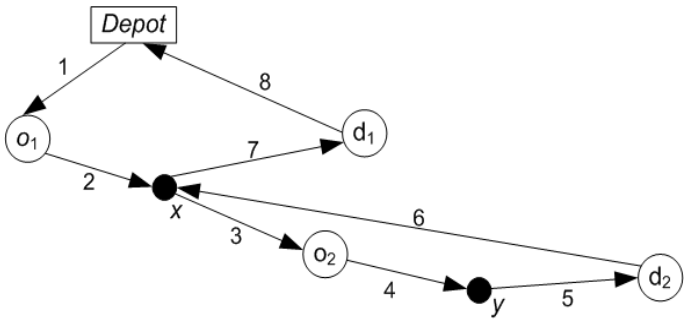


FIGURE 3. Visualizing a valid tour (arc labels indicate the order according to which arcs are visited).

- start(First(Γ)) = End(Last(Γ)) = *Depot;*
- any node x in $X_O \cup X_D$ is involved in exactly two labelled links r and $r' = \text{Succ}(\Gamma, r)$: this means that V moves to $o_k(d_k)$, $k \in K$, only when it comes to start (finish) dealing with demand k;
- the *Depot* node is involved only in both labelled links $r = \text{First}(\Gamma)$ and $r' = \text{Last}(\Gamma)$;
- for any demand $k \in K$, the deriving subsequence Γ^k related to k is such that:
  - $\text{Start}(\text{First}(\Gamma^k)\ ) = o_k$;
  - $\text{End}(\text{Last}(\Gamma^k)) = d_k$;
  - For any consecutive labelled link pair r, $r' = \text{Succ}(\Gamma^k, r)$, we have $\text{End}(r) = \text{Start}(r')$.

Figure 3 visualizes the valid tour $\Gamma = \{(\textit{Depot}, o_1, 0), (o_1, x, 1), (x, o_2, 0), (o_2, y, 2), (y, d_2, 2), (d_2, x, 0), (x, \text{d}_1, 1), (\text{d}_1, \textit{Depot}, 0)\}$.

Then the **APSCP** may be set as follows: {*given the node set X and a shortest path distance matrix D that means a* $X \cdot X$ *distance matrix which satisfies triangle inequality, compute a valid tour* Γ *with minimum cost*}.

3. A TREE REPRESENTATION OF TOURS

The goal of this section is to describe the way some specific tours can be represented by trees. This representation will integrate both the preemption hypothesis and the vehicle capacity constraint. So dealing with APSCP while restricting ourselves to those specific tours will mean solving a kind of unconstrained problem and can be done through both efficient and simple tree handling algorithms. Of course it will not be possible to represent every tour according to such a tree representation. But the *restriction theorem* of Section 3.1. is going to make clear that, in any case, an optimal solution of APSCP may be found among those tours which admit this tree representation.

3.1. The restriction theorem

In order to enable us to correctly state this *restriction theorem* we need to introduce some additional definitions and notations. Let Γ be a valid tour. For any labelled link $r = (x,\ o_k,\ 0)$ in Γ, we denote by $\sigma(\Gamma,\ r)$ the unique labelled link $= (d_k, y, 0)$ which is also in Γ. So r is related to the first arrival of the vehicle V in o_k and $\sigma(\Gamma,\ r)$ is related to the last departure of V from $d_k V$ being empty in both cases. In the same way, if $r = (y,\ x,\ k)$, $k \neq 0$, is some labelled link in Γ such that x is a reload node in X_R, then we also denote by $\sigma(\Gamma,\ r)$ the first triple $r' = (x,\ z,\ k)$ which is in Γ^k Thus r is related to an arrival of V at the reload node x and r' is related to a departure of V from x, carrying V in both cases, the demand k load One may notice that $\sigma(\Gamma,\ r)$ is not defined for all labelled links r.

We say that two labelled links r and r' in Γ are *overlapping* if $\sigma(\Gamma,\ r')$ and $\sigma(\Gamma,\ r)$ are defined and if they are such that:

$$\text{Rank}(\Gamma, \sigma(\Gamma,\ r')) > \text{Rank}(\Gamma, \sigma(\Gamma,\ r)) >\ \text{Rank}(\Gamma,\ r') > \text{Rank}(\Gamma,\ r).$$

Overlapping labelled links corresponds to demand pairs k and k' which are such that o_k, $o_{k'}$, d_k, $d_{k'}$ are visited according to this order. We shall see further that applying the restriction theorem will consist in avoiding the appearance of such overlapping links inside an optimal tour.

Then we may state the following restriction theorem, whose meaning is that one may restrict the search for an optimal solution of **APSCP** to a sub-domain whose elements are valid tours endowed with additional properties which will allow us to code them in a tree design.

Restriction theorem.

Let Γ be some optimal tour for the **APSCP** problem, which we suppose chosen in such a way that:

- (A) $|\Gamma|$is the smallest possible;
- (B) the number of labelled links r in the tour Γ, for which (A) is supposed to be satisfied which are such that $\text{Label}(r) \neq 0$ is the smallest possible;

Then, the following assertions must be true:

- (S1) Γ does not contain two occurrences of the same labelled link $r = (x,\ y,\ k)$, with $k \neq 0$;
- (S2) Γ does not contain two consecutive labelled links r and r' such that $\text{Label}(r) = \text{Label}(r')$;
- (S3) Γ does not contain two overlapping labelled links r and r';
- (S4) Γ does not contain two labelled links r and r' such that $\text{End}(r) = \text{End}(r')$ and both $\text{Label}(r)$, $\text{Label}(r') \neq 0$.

In other words (S1) means that the same commodity is never carried twice in the same link; (S2) means that every time the vehicle reaches a node it is in order to perform either a load action or an unload action; (S3) means that if the vehicle unloads a demand k at a reload node x it must not take back this demand k before having delivered all the demands it has loaded after the unload of k; (S4) means that the vehicle does not reaches twice the same node while being not empty. Since the proof of our restriction theorem is going to be divided into four

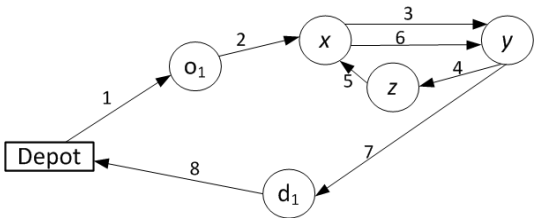


Figure 4 represents the tour $\{(Depot, o_1, 0), (o_1, x, 1), (x, y, 1), (y, z, 1), (z, x, 1), (x, y, 1), (y, d_1, 1), (d_1, Depot, 0)\}$. The link$r = (x, y, 1)$ appears twice.

FIGURE 4. A tour with the configuration prohibited by (S1) (arc labels indicate the order according to which arcs are visited).

parts, respectively related to (S1) ... (S4) we shall include in those four parts new Figures 4–6 and 8 representing configurations that are prohibited by these four assertions.

Proof.

We assume that Γ is given, which is an optimal solution of **APSCP** and which is such that (A) and (B) are true.

Part (S1).

If $r = (x, y, k)$, $k > 0$ appears twice in Γ, with respectively rank s and s', then x and y are both reload nodes, and we may replace by 0 the label value Label(r'') in any labelled link r'' which is such that:

- $s \leqslant \text{Rank}(\Gamma, r'') < s'$;
- Label$(r'') = k$

While doing it, we keep on with a valid tour which is an optimal solution of **APSCP** and we get a contradiction on the (B) hypothesis. Figure 4 shows a tour in which the labelled link $r = (x, y, 1)$ appears twice.

Part (S2).

If $r = (x, y, k)$ and $r' = (y, z, k)$ were two consecutive labelled links of Γ such that Label$(r) =$ Label$(r') = k$, then we would be able to remove both r and r' from Γ, and replace them by a unique labelled link (x, z, k). While doing this, we would also keep, because of the triangle property on the distances, an optimal solution of **APSCP**, and this solution would contradict the (A) hypothesis. Figure 5 shows a tour in which two consecutive labelled links have the same label.

Part (S3).

Let us suppose that there exist two labelled links $r = (y, x, k)$ and $r' = (y', x', k')$ which are overlapping in Γ. We may choose them in such a way that Rank$(\Gamma, \sigma(\Gamma, r'))$ – Rank(Γ, r) is the smallest possible (E1). Because of (E1), we see that x and x' must be reload nodes: if, for instance, x were an origin node o_h, then there would exist a labelled link r'' with Label$(r'') = h$ and $h \neq k$ such that:

Rank$(\Gamma, r) <$ Rank$(\Gamma, r'') <$ Rank$(\Gamma, r') <$ Rank$(\Gamma, \sigma(\Gamma, r'')) <$ Rank$(\Gamma, \sigma(\Gamma, r)) <$ Rank$(\Gamma, \sigma(\Gamma, r'))$.

Then we might deduce a new overlapping pair (r'', r') which would induce a contradiction on the minimality assumption (E1). In the same way, we may check

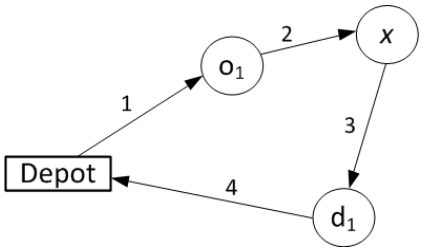


Figure 5 represents the tour $\{(Depot, o_1, 0), (o_1, x, 1), (x, d_1, 1), (d_1, Depot, 0)\}$. The two consecutive links $(o_1, x, 1)$ and $(x, d_1, 1)$ can be replaced by $(o_1, d_1, 1)$: the tour resulting is clearly better due to triangle property on the distances.

FIGURE 5. A tour with the configuration prohibited by (S2) (arc labels indicate the order according to which arcs are visited).

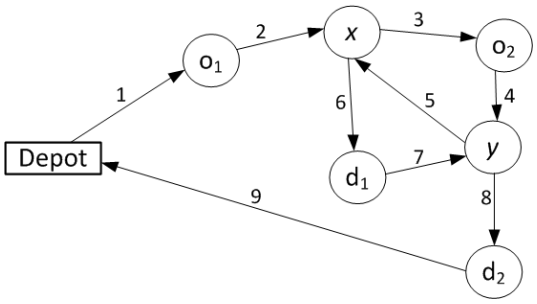


Figure 6 represents the tour $\Gamma = \{(Depot, o_1, 0), (o_1, x, 1), (x, o_2, 0), (o_2, y, 2), (y, x, 0), (x, d_1, 1), (d_1, y, 0), (y, d_2, 2), (d_2, Depot, 0)\}$. The two labelled links $r = (o_1, x, 1)$ and $r' = (o_2, y, 2)$ are overlapping since: $\text{Rank}(r) < \text{Rank}(r') < \text{Rank}(\sigma(\Gamma, r)) < \text{Rank}(\sigma(\Gamma, r))$ where $\sigma(\Gamma, r) = (x, d_1, 1)$ and $\sigma(\Gamma, r') = (y, d_2, 2)$.

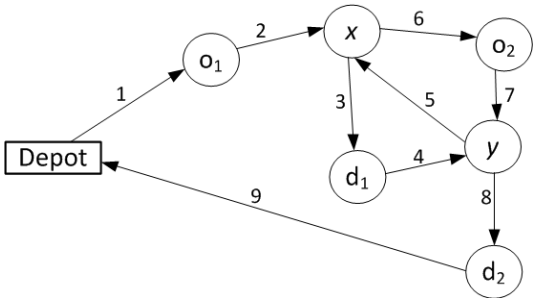
FIGURE 6. A tour with two overlapping labelled links (arc labels indicate the order according to which arcs are visited).

that x' cannot be an origin node o_h. Thus x and x' are both reloads nodes, and we clearly have: $k \neq k' \neq 0$.

It comes that we may write:

- $r = (y, x, k)$ and $r' = (y', x', k')$;
- $\sigma(\Gamma, r) = (x, z, k)$ and $\sigma(\Gamma, r') = (x', z', k')$.

So, we set :
$I_1 = I(\Gamma, \text{First}(\Gamma), r)$;
$I_2 = I(\Gamma, \text{Succ}(\Gamma, r), r')$;
$I_3 = I(\Gamma, \text{Succ}(\Gamma, r'), \text{Pred}(\Gamma, \sigma(\Gamma, r)))$;
$I_4 = I(\Gamma, \sigma(\Gamma, r), \text{Pred}(\Gamma, \sigma(\Gamma, r')))$;
$I_5 = I(\Gamma, \sigma(\Gamma, r'), \text{Last}(\Gamma))$;
and we replace Γ by the concatenation $\Gamma_{\text{Aux}} = I_1 \oplus I_4 \oplus I_3 \oplus I_2 \oplus I_5$. Of course, the lengths of Γ and Γ_{Aux} are equal, as well as their respective costs. Moreover Γ_{Aux} defined above is a valid tour. Indeed, we only need to check that switching I_2, I_3 and I_4 does not break any sequence Γ^k, or, in other words, that for any $k \neq 0$, we have $\Gamma^k = \Gamma^k_{\text{Aux}}$. If the converse were true, we would be able to find $k'' \neq k'$, k, $k'' \neq 0$, as well as two labelled links r'' and $\sigma(\Gamma, r'')$, with label k''



According to the notations in the proof of (S3):
$I_1 = \{(Depot, o_1, 0), (o_1, x, 1)\}$
$I_2 = \{(x, o_2, 0), (o_2, y, 2)\}$
$I_3 = \{(y, x, 0)\}$
$I_4 = \{(x, d_1, 1), (d_1, y, 0)\}$
$I_5 = \{(y, d_2, 2), (d_2, Depot, 0)\}$
This figure represents the final tour $\Gamma_{\text{Aux}} = \{(Depot, o_1, 0), (o_1, x, 1), (x, d_1, 1), (d_1, y, 0), (y, x, 0), (x, o_2, 0), (o_2, y, 2), (y, d_2, 2), (d_2, Depot, 0)\}$.

FIGURE 7. The tour resulting from the transformation of the one depicted in Figure 6 (arc labels indicate the order according to which arcs are visited).

or with the ending node of r'' equal to $o_{k''}$, in such a way that one of the three following relations would be true:

- $r'' \in I_2$ and $\sigma(\Gamma, r'') \in I_3$; (E2)
- $r'' \in I_2$ and $\sigma(\Gamma, r'') \in I_4$; (E3)
- $r'' \in I_3$ and $\sigma(\Gamma, r'') \in I_4$. (E4)

In case (E2) or (E3) were true, r'' and r' would be overlapping, and would contradict the (E1) hypothesis, related to the minimality of $\text{Rank}(\Gamma, \sigma(\Gamma, r')) - \text{Rank}(\Gamma, r)$.

In case (E4) were true, r and r'' would be overlapping, and would contradict the (E1) hypothesis, related to the minimality of $\text{Rank}(\Gamma, \sigma(\Gamma, r')) - \text{Rank}(\Gamma, r)$.

So, in any case, Γ_{Aux} is a valid tour.

This result allows us to conclude the proof of (S3) by noticing that r and $\sigma(\Gamma, r)$ become consecutive in the valid tour Γ_{Aux}, which implies (proof of statement (S2)) that r and $\sigma(\Gamma, r)$ may be replaced in Γ_{Aux} by a unique labelled link (y, z, k) in such a way that $\text{Cost}(\Gamma_{\text{Aux}})$ does not increase and that $|\Gamma_{\text{Aux}}|$ decreases, inducing a contradiction on the (A) hypothesis. This part of proof is illustrated by Figures 6 and 7: Figure 6 shows a tour Γ with two overlapping labelled links while Figure 7 shows the tour Γ_{Aux} resulting from the transformation proposed.

Part (S4)

Let us suppose that Γ contains two labelled links r and r' such that $\text{End}(r) = \text{End}(r')$ and such that $\text{Rank}(\Gamma, r) < \text{Rank}(\Gamma, r')$. Since the starting node x of $\sigma(\Gamma, r)$ cannot be in $\{Depot\} \cup X_D$ it must belong to X_R and must be used twice as a reload node. So, r, $\sigma(\Gamma, r)$, r' and $\sigma(\Gamma, r')$ may be written:

- $r = (y, x, k)$, $k \neq 0$ and $r' = (y', x, k')$, $k' \neq 0$, k;
- $\sigma(\Gamma, \text{r}) = (x, z, k)$ and $\sigma(\Gamma, r') = (x, z', k')$.

Because of (S3) we must have:

- $\text{Rank}(\Gamma, r) < \text{Rank}(\Gamma, \sigma(\Gamma, r)) < \text{Rank}(\Gamma, r') < \text{Rank}(\Gamma, \sigma(\Gamma, r'))$ (E5)

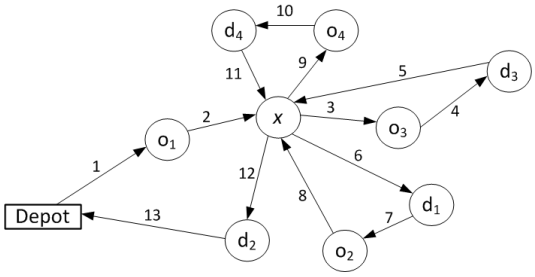


Figure 8 represents the tour $\Gamma = \{(Depot, o_1, 0), (o_1, x, 1), (x, o_3, 0), (o_3, d_3, 3), (d_3, x, 3), (x, d_1, 1), (d_1, o_2, 0), (o_2, x, 2), (x, o_4, 0), (o_4, d_4, 4), (d_4, x, 0), (x, d_2, 2), (d_2, Depot, 0)\}$.
The two labelled links $r = (o_1, x, 1)$ and $r' = (o_2, x, 2)$ are such that $\text{End}(r) = \text{End}(r')$ and both $\text{Label}(r), \text{Label}(r') \neq 0$.

FIGURE 8. A tour including the configuration prohibited by (S4) (arc labels indicate the order according to which arcs are visited).

or

- $\text{Rank}(\Gamma, r) < \text{Rank}(\Gamma, r') < \text{Rank}(\Gamma\sigma(\Gamma, r')) < \text{Rank}(\Gamma, \sigma(\Gamma, r))$. (E6)

Let us first suppose that (E5) holds. Then we set:

$I_1 = I(\Gamma, \text{First}(\Gamma), r)$;
$I_2 = I(\Gamma, \text{Succ}(\Gamma, r), \text{Pred}(\Gamma, \sigma(\Gamma, r)))$;
$I_3 = I(\Gamma, \sigma(\Gamma, r), r')$;
$I_4 = I(\Gamma\ \text{Succ}(\Gamma, r'), \text{Pred}(\Gamma, \sigma(\Gamma, r')))$;
$I_5 = \text{I}, (\Gamma, \sigma(\Gamma, r'), \text{Last}(\Gamma))$;

and we replace Γ by the concatenation $\Gamma_{\text{Aux}} = I_1 \oplus I_3 \oplus I_2 \oplus I_4 \oplus I_5$. Of course, the lengths of Γ and Γ_{Aux} are equal, as well as their respective costs, and we proceed as in the proof of (S3) in order to prove that Γ_{Aux} must be a valid tour. But we also notice as in the proof of (S3), that Γ_{Aux} can be shortened by replacing the consecutive labelled links r and r' by a unique labelled link (y, z, k), in such a way that $\text{Cost}(\Gamma_{\text{Aux}})$ does not increase and that $|\Gamma_{\text{Aux}}|$decreases, inducing a contradiction on the (A) hypothesis.

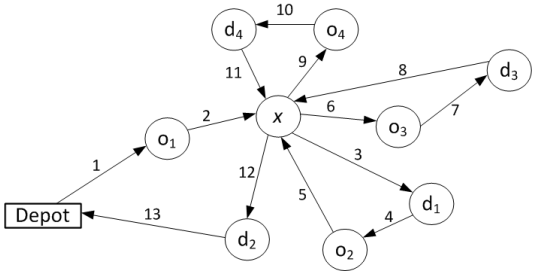
We apply exactly the same kind of reasoning in case (E6) holds. This part of proof is illustrated by Figures 8 and 9: Figure 8 shows a tour Γ with two labelled links r and r'such that $\text{End}(r) = \text{End}(r')$ and both $\text{Label}(r), \text{Label}(r') \neq 0$ while Figure 9 shows the tour Γ_{Aux} resulting from the transformation proposed.

3.2. A TREE REPRESENTATION OF THE APSCP PROBLEM

Restriction theorem leads us to restrict our search, when we deal with the **APSCP**, to valid tours which satisfy the above (S1)...(S4) properties. We will call such valid tour *strongly valid tour* Figure 10 shows us how the valid tour Γ of Figure 3 may be turned into a strongly valid tour without any cost loss.

Clearly, solving the APSCP Problem means finding a strongly valid tour Γ with minimum cost value.

Now, we are going to see that any strongly valid tour may be represented as a particular tree, and this will provide us with the basis for the algorithmic handling of **APSCP** which will be proposed in Section 4.



According to the notations in the proof of (S4):
$I_1 = \{(Depot,\ o_1,\ 0),\ (o_1,\ x,\ 1)\} I_2 = \{(x,\ o_3,\ 0),\ (o_3,\ d_3,\ 3),\ (d_3,\ x,\ 3)\}$
$I_3 = \{(x,\ d_1,\ 1),\ (d_1,\ o_2,\ 0),\ (o_2,\ x,\ 2)\}$
$I_4 = \{(x,\ o_4,\ 0),\ (o_4,\ d_4,\ 4),\ (d_4,\ x,\ 0)\}$
$I_5 = \{(x,\ d_2,\ 2),\ (d_2,\ Depot,\ 0)\}$.
And so this figure represents the final tour $\Gamma_{\text{Aux}} = \{(Depot, o_1, 0),\ (o_1,\ x,\ 1),\ (x,\ d_1,\ 1),\ (d_1,\ o_2,\ 0),\ (o_2,\ x,\ 2)\ (x,\ o_3,\ 0),\ (o_3,\ d_3,\ 3),\ (d_3,\ x,\ 3),\ (x,\ o_4,\ 0),\ (o_4,\ d_4,\ 4),\ (d_4,\ x,\ 0),\ (x,\ d_2,\ 2),\ (d_2,\ Depot,\ 0)\}$.

FIGURE 9. The tour resulting from the transformation of the one depicted in Figure 8 (arc labels indicate the order according to which arcs are visited).

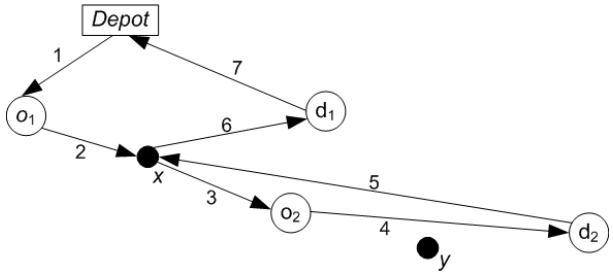


FIGURE 10. A derivation of the valid tour Γ of Figure 1 into a strongly valid tour Γ'.

Bipartite ordered trees: given two classes of nodes A and B we say that a subgraph T of $G(A \cup B,\ A \times B)$ is a *bipartite ordered tree* if:

- nodes in class A have their sons in class B and conversely;
- for every node x in T which is not a terminal node (leaf), the son set associated with x is linearly ordered and is consequently described as a sequence.

Remark: The "ordered" term means that the way the sons of a given node are ordered will have an impact on the semantics of the tree, that is to say on the way an ordered tree is going to be interpreted as a valid APSCP tour.

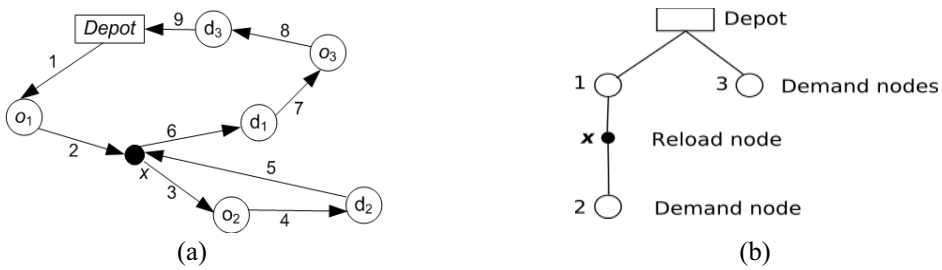


FIGURE 11. A strongly valid tour Γ; (b) the related bipartite tree Tree(Γ).

In order to explain what we expected by the bipartite ordered tree concept, let us considered a tour Γ which satisfied the (S1)$\cdots$(S4) properties and let us deduce from it a bipartite ordered tree Tree(Γ).

Figure 11 features a strongly valid tour Γ and the bipartite tree T = Tree(Γ) which derives from this tour. We can describe in an informal way the construction of the tree as follows:

(1) the vehicle V starts at the depot: so the depot is the root of T;
(2) V goes to the origin node o_1: so node *demand 1* becomes a son of depot node in T;
(3) V unloads the unit load of the demand 1 at reload node x: so reload node x becomes a son of node *demand 1* in T;
(4) V satisfies the demand 2 and goes back to x: *so* node *demand 2* is the unique son of reload node x in T;
(5) after reloading the unit load of the demand 1 in x and deliver it in d_1, V satisfies directly (without any reload) the demand 3: so node *demand 3* is the second son of depot.
(6) V goes back to the depot: the construction of T is then finished.

We see that the nodes of the tree T = Tree(Γ) are alternatively *demand nodes* and *reload nodes*. In order to describe this construction in a more formal way we need to state which properties must be satisfied by a bipartite tree T in order to make possible the construction of a tour Γ such that T = Tree(Γ) Given an **APSCP** instance defined by the demand set K and by the node set X, this leads us to define a (X, K)-*consistent* bipartite ordered tree T as a bipartite ordered tree T such that:

- a node in T can be identified either with a demand $k \in K$ (we shall then talk about *demand* node) or with a node in $\{Depot\} \cup X_R$, (and then we talk about *reload* node); any possible demand node $k \in K$ appears in T, while only some of nodes of $\{Depot\} \cup X_R$ appear in T: those nodes in $\{Depot\} \cup X_R$ which appear in T define the *active* reload node set ACTIVE(T) of T; (S5)
- the root of T is the *Depot* node and the terminal nodes (leafs) of T are all demand nodes; (S6)

- for any demand node k, its linearly ordered *son* set RELOAD(T, k) (which may be empty) is made with active reload nodes and its father FATHER(T, k) is in ACTIVE(T); (S7)
- for any reload node x, its linearly ordered *son* set DEMAND(T, x) is made with demand nodes and its father FATHER(T, x) is in K. (S8)

Tree*(Γ) *construction

This allows us to describe in a formal way the intuitive construction of Figure 11: given a strongly valid tour Γ we get $T = \text{Tree}(\Gamma)$ from Γ as follows:

- ACTIVE(T) is defined as the set of the nodes of $\{Depot\} \cup X_R$ which appear in some labelled link of Γ, and which are then said to be the *active reload* nodes for Γ;
- for any demand node $k \in K$, the son set Reload(T, k) is made with the reload nodes which appear in some labelled link of Γ^k, ordered according to their appearance order in Γ^k;
- for any active reload node x in $\{Depot\} \cup X_R$, we denote by $\rho(x) = (x, y, 0)$ and by $\tau(x) = (z, x, 0)$ the two labelled links with label 0 which involve x in Γ and which are such that: $\text{Rank}(\Gamma, \rho(x)) < \text{Rank}(\Gamma, \tau(x))$. Then we define the son set Demand(T, x) by setting that a demand $k \in K$ is a son of x if the unique labelled link $r(k) = (o_k, t, k)$, t in X which appears in Γ is such that:
  - $\text{Rank}(\Gamma, \rho(x)) < \text{Rank}(\Gamma, r(k)) < \text{Rank}(\Gamma, \tau(x))$;
  - there exists no reload node y such that $\text{Rank}(\Gamma, \rho(x)) < \text{Rank}(\Gamma, \rho(y)) < \text{Rank}(\Gamma, \text{r}(k)) < \text{Rank}(\Gamma, \tau(y)) < \text{Rank}(\Gamma, \tau(x))$.

TreeCost value of a consistent bipartite ordered tree

For any (X, K)-consistent bipartite ordered tree T, we define a cost value TreeCost(T) as follows:

- for any demand node $k \in K$, we set:

If k is a terminal node then

$$\text{CDem}\,(T, k) = D_{o_k, d_k}$$

else

$$\text{CDem}(T, k) = D_{o_k, \text{First}(\text{Reload}(T,k))} + D_{\text{Last}(\text{Reload}(T,k)), d_k} + \sum_{\substack{x \text{Reload}\,(T,\,k)\,, \\ x \neq \text{ Last }(\text{Reload }(T,k))}} D_{x, \text{Succ}(\text{Reload}(T,k),x)}$$

- for any reload node $x \in \{Depot\} \cup X_R$, we set:

$$\text{CRel}\,(T, x) = D_{x, o_{\text{First}(\text{Demand}(T,x))}} + D_{d_{\text{Last}(\text{Demand}(T,x))}, x} + \sum_{\substack{k \,\in\, \text{Demand}\,(T,x) \\ k \neq \text{ Last}(\text{Demand}\,(T,\,x)\,)}} D_{d_k, o_{\text{Succ}(\text{Demand}(T,x),k)}}.$$

Finally,

$$\text{TreeCost}\,(T) = \sum_{k \in K} \text{CDem}\,(T, k) + \sum_{x \in \text{ACTIVE}(T)} \text{CRel}\,(T, x).$$

Then the following results are going to turn the **APSCP** problem into a somewhat simpler problem related to the search for specific trees.

Tree representation theorem

The above described *Tree* correspondence is a one-to-one correspondence between the strongly valid tours and the (X, K)-consistent bipartite ordered trees; for any such a tour Γ, we have: $\text{TreeCost}(\text{Tree}(\Gamma)) = \text{Cost}(\Gamma)$.

Proof.

(1) The equality $\text{TreeCost}(\text{Tree}(\Gamma)) = \text{Cost}(\Gamma)$ comes in an easy way by construction;

(2) we check that *Tree* is a one-to-one correspondence:

- Tree is injective: let Γ and Γ' two different valid tours such that $\Gamma \neq \Gamma'$ then $\text{Tree}(\Gamma) \neq \text{Tree}(\Gamma')$. Indeed we scan Γ and Γ' until they differ, and we see that the above construction make $\text{Tree}(\Gamma)$ and $\text{Tree}(\Gamma')$ be different.
- Tree is onto: Let a tree T, then it exists a tour Γ such that $T = \text{Tree}(\Gamma)$ Indeed, to get Γ we only have to perform a depth first search of the tree T.

Then it comes that the so defined ***Tree*** correspondence is as it is claimed in the statement of the tree representation theorem □

We deduce from the tree representation theorem the following corollary:

Corollary 3.1. *Solving an **APSCP** instance (X, K) means finding a bipartite ordered tree T consistent with (X, K) such that* TreeCost*(T) is the smallest possible.*

The interest of this last statement is clearly that it provides us with a bipartite tree formulation of the **APSCP** problem which is far less constrained that the original one. Moreover it is going to provide us with an ILP (integer linear programming) formulation of APSCP.

3.3. An integer linear programming (ILP) formulation of APSCP

An ILP formulation of APSCP may be derived from the previous structural results. In order to get it, we first build the following auxiliary network $G = (X^*, E)$:

- $X^* = X \cup X_R^* \cup \{Depot^*\}$, where X_R^* is a copy of X_R and $Depot^*$ is a copy of *Depot*.

For any node x in X_R, we denote by x^* its copy in X_R^*. In a similar way, for any origin node $x = o_k$ in X_O, we denote by x^* the related node d_k in X_D.

- $E = \{(Depot, x), x \in X_O\} \cup \{(x, Depot^*), x \in X_D\} \cup \{(o_k, d_k), k \in K\} \cup \{(d_{k'}, o_k), k \neq k' \in K\} \cup \{(x, y), (y, x), x \in X_O, y \in X_R\} \cup \{(x, y)\ (y, x), x \in X_R^*, y \in X_D\} \cup \{(x, y), x \in X_R^*, y \in X_R\}$.

Every arc e in E is then provided with a length D_e^* which derives from the D distance matrix in a natural way.

Let us recall that a path γ of a such a network G is a node sequence γ such that, for any node x in γ, the pair $(x, \mathrm{Succ}(\gamma, x))$ defines an arc of E. One easily checks that any strongly valid tour Γ can be turned into a path Γ^* of the network G, in such a way that:

- (S9) Γ^* starts from *Depot* and ends into $Depot^*$ and Γ^* is an elementary path, *i.e.*, its visits any node at most once;
- (S10) for every k in K, Γ^* visits o_k and d_k according to this order, and for every x in X_R, Γ^* visits x if and only if it visits x^*, and, in this case, it does it according to this order;
- (S11) for any pair $x, y, x \neq y$, in $X_R \cup X_O$ the following implication is true:

$$(\mathrm{Rank}(\Gamma^*, x) < \mathrm{Rank}(\Gamma^*, y) \textbf{ and } \mathrm{Rank}(\Gamma^*, y) < \mathrm{Rank}(\Gamma^*, x^*)) => \mathrm{Rank}(\Gamma^*, y^*) < \mathrm{Rank}(\Gamma^*, x^*).$$

We call this condition the *non overlapping condition*

- (S12) $\mathrm{Cost}(\Gamma) = \sum_{\substack{x \in \Gamma^* \\ x \neq Depot^*}} \mathrm{D}^*_{x, Succ(\Gamma^*, x)}$

So we say that a path of the network G which satisfies (S9)...(S12) above is a *strongly valid path.*

Theorem 3.2. *For any strongly valid path γ, there exists a strongly valid tour Γ such that $\Gamma^* = \gamma$.*

Proof. Let us first describe the way Γ is going to derives from γ. It will occur through the following **RECONSTRUCT** procedure:

RECONSTRUCT Procedure

$x \leftarrow Depot$;
$\Gamma \leftarrow Nil$;
While (x $\neq Depot^*$) **do**
$y \leftarrow \mathrm{Succ}(\gamma, x)$;
If (y may be written $y = z^*$, with $z \in \{Depot\} \cup X_R$, **then**
we set $\Pi(y) = z$,
else
we set $\Pi(y) = y$;
If (multi-case branching instruction)
1. $x = Depot$ **then** $\Gamma \leftarrow \{(x, y, 0)\} \oplus \Gamma$;
2. x $= o_k$, $k \in K$ **then** $\Gamma \leftarrow \{(x, y, \mathrm{k})\} \oplus \Gamma$;
3. $x \in X_R$ **then** $\Gamma \leftarrow \{(x, \Pi(y), 0)\} \oplus \Gamma$;

4. $x \in X_D$ **then** $\Gamma \leftarrow \{(x, \Pi(y), 0)\} \oplus \Gamma$;
5. $x \in X_R^*$ **then** $\Gamma \leftarrow \{(x, y, k)\} \oplus \Gamma$, where $k \neq 0$ is such that the labelled link
$(\text{Pred}(\gamma, \Pi(x)), \Pi(x), k)$ is already in Γ; (I1)

The instruction (I1) works here because of (S10) above, and because an arc of G which arrives on $\Pi(x)$ must come from an origin node o_k or from a node z in X_R^*. In this last case, a simple induction reasoning makes appear the fact that k is different from 0.

We get our result while proceeding by induction on the length (the number of nodes) of γ. In case γ involves no node in X_R, then the results comes in a trivial way. Else, we consider $x \in X_R$ which is the first node of X_R which appears in γ. We notice that $\text{Pred}(\gamma, x_0)$ must be some node o_k, $k \in K$. Thus the arc $(\text{Pred}(\gamma, x_0), \text{Succ}(\gamma, x_0^*))$ belongs to the arc set E, and the removal of the subpath $\text{I}(\gamma, x_0, x_0^*)$ from γ provides us with another path γ_1 of the graph G. Let us set:

- $K_1 = \{k \in K$ such that o_k and d_k are nodes of $\gamma_1\}$;
- $K_2 = \{k \in K$ such that o_k and d_k are nodes of $\gamma_2 = \text{I}(\gamma, x_0, x_0^*)\}$;

K_1 and K_2 define a partition of K, and one sees that γ_2 may be viewed as a strongly valid path, if we restrict ourselves to K_2 as a demand set and if we consider that x_0 and x_0^* play the role of *Depot* and *Depot**. Thus it comes from the induction hypothesis that it may also be written, under this restriction, according to the form $\gamma_2 = \Gamma_2^*$. By the same way, γ_1 is also a strongly valid path if we restrict the demand node set to K_1, and it comes from the induction hypothesis that it may be written, under this restriction, according to the form $\gamma_1 = \Gamma_1^*$. We only need to insert Γ_2 between $(\text{Pred}(\gamma, x_0), x_0, k)$ and $(x_0, \text{Succ}(\gamma, x_0^*), k)$ in Γ_1 in order to get Γ such that $\gamma = \Gamma_0^*$. □

Corollary 3.3. *Solving a* ***APSCP*** *instance* (X, D, K) *means finding a strongly valid path* γ *with minimal length (for the* D^* *length function) value in the network* G.

Since it will be possible to represent any strongly valid path as a combination of a $\{0, 1\}$-flow vector z defined on the network G, of a rank vector R defined on the vertex set of G, and of an auxiliary $\{0, 1\}$-decision vector t dedicated to the expression of the properties (S1)…(S4), we become able to state:

Theorem 3.4. *Solving an* ***APSCP*** *instance* (X, D, K) *means solving the integer linear program* (P) *in which unknown vectors are:*

- $z = (z_e, e \in E)$, *with values in* $\{0, 1\}$;
- $R = (R_x, x \in X)$ *integer and positive;*
- $t = (t_{x,y}, x \neq y, x, y \in X_R \cup X_O)$ *with values in* $\{0, 1\}$.

$$(P)\quad \begin{cases} \text{Minimize } \sum\limits_{e\in E} DIST^*(e)\cdot z_e & \\ \sum\limits_{x\in X^*} z_{(x,Depot)} = 0 & (C1)\\ \sum\limits_{x\in X^*} z_{(x,Depot^*)} = 1 & (C2)\\ \sum\limits_{x\in X^*} z_{(Depot,x)} = 1 & (C3)\\ \sum\limits_{x\in X^*} z_{(Depot^*,x)} = 0 & (C4)\\ \forall x\in X_D\cup X_O,\ \sum\limits_{y\in X^*} z_{(y,x)} = 1 & (C5)\\ \forall x\in X_R\cup X_{R^*},\ \sum\limits_{y\in X^*} z_{(y,x)} \leqslant 1 & (C6)\\ \forall x\in X_R,\ \sum\limits_{y\in X^*} z_{(y,x)} = \sum\limits_{y\in X^*} z_{(y,x^*)} & (C7)\\ \forall k\in K, R_{o_k} \leqslant R_{d_k} - 1 & (C8)\\ \forall x\in X_R, R_x \leqslant R_{x^*} - 1 & (C9)\\ \forall x\in X^* - \{Depot, Depot^*\},\ \sum\limits_{y\in X^*} z_{(y,x)} = \sum\limits_{y\in X^*} z_{(x,y)} & (C10)\\ \forall (x,y)\in E,\ z_{(x,y)} + \frac{R_x + 1 - R_y}{|X^*|} \leqslant 1 & (C11)\\ \forall (x,y), x\neq y, x, y\in X_R\cup X_O,\ t_{xy} + \frac{R_y + 1 - R_{x^*}}{|X^*|} \leqslant 1 & (C12)\\ \forall (x,y), x\neq y, x, y\in X_R\cup X_O,\ t_{xy} + \frac{R_y - 1 - R_{x^*}}{|X^*|} \geqslant 0 & (C13)\\ \forall (x,y), x\neq y, x, y\in X_R\cup X_O,\ t_{xy} + \frac{R_{y^*} + 1 - R_{x^*}}{|X^*|} \leqslant 1 & (C14) \end{cases}$$

The constraints of (P) *have the following meaning:*

- $(C1)\ldots(C4)$: *the inflow induced by* z in $Depot\,(Depot^*)$ *is equal to 0 (1), while the related outflow is equal to 1 (0);*
- $(C5)$: *in any node of* $X_D\cup X_O$*, the inflow induced by z is equal to 1 (translation of (S9) and (S10));*
- $(C6)$, $(C7)$: *in any node of* $X_R\cup X_{R^*}$*, the inflow induced by z is at most equal to 1, and the inflow value in x is equal to the inflow value in* x^* *(translation of (S10) and (S9));*
- $(C8)$: *the rank of a demand origin node is lower than the rank of the related destination node (translation of (S10));*
- $(C9)$: *the rank of a reload node x is lower than the rank of the related reload node* x^* *(translation of (S10));*
- $(C10)$: *z is a flow vector, which satisfies the usual Kirshoff law in any node but in Depot and* $Depot^*$*;*
- $(C11)$: *translation of the implication* $z_e = 1\,[ERR : mgroupChr : nParams = 0, isTop = 1, iOp = 0x21D2]\,R_x + 1 - R_y \leqslant 0$ *;*
- $(C12)\ldots(C14)$: *translation of the non overlapping condition.*

Proof. The above model involves a {0,1}-vector flow $z = (z_e,\ e \in E)$, a rank integer vector $R = (R_x,\ x \in X)$, as well as a *positional* $\{0,\ 1\}$ vector t, which is indexed on the pairs $(x,\ y)$, $x \neq y$, $x,\ y \in X_R \cup X_O$, with the following semantics:

- for any arc e in E, $z_e = 1$ iff the arc e is in the strongly valid path γ;
- for any node x in γ, R_x will provide us with the rank of x in γ;
- for any pair $(x,\ y)$, $x \neq y$, $x,\ y \in X_R \cup X_O$:
  - $t_{x,y} = 1$ iff $\text{Rank}(\gamma,\ y) < \text{Rank}(\gamma,\ x^*)$, *i.e.* y is located before x^* in the tour defined by z;
  - $t_{x,y} = 0$ iff $\text{Rank}(\gamma,\ x^*) < \text{Rank}(\gamma,\ y)$, *i.e.* x^* is located before y in the tour defined by z

Then getting our result is only a matter of applying corollary 2 and of translating into the ILP formalism the (S9) ... (S12) requirements. Namely, we check that, by simultaneously playing with $t_{x,y}$ and $t_{y,x}$, one get that the last constraints of the above linear program allows us to keep the valid tour defined by z to admit any overlapping pair of labelled links. □

4. Tree based heuristics for the APSCP problem

The algorithms which we are going to describe and test here, derive in a straightforward way from the tree representation of the **APSCP** Problem which we got in Section 3. These algorithms are simple greedy insertion algorithms and descent algorithms, based upon the use of 2 classes of operators: insertion operators and local transformation operators

4.1. Greedy insertion algorithm

Greedy insertion algorithms are designed in order to provide an initial solution. They use insertion operators which act on some bipartite ordered tree T consistent with the node set X and with a subset K' of the demand set K, and *insert* some demand $k \in K - K'$ into T. We use two operators:

- INSERT-SIMPLE: its parameters are some active reload node x in $\{Depot\} \cup X_R$, and some cut $(l_1,\ l_2)$ of the sequence $\text{DEMAND}(T,\ x) = l_1 \oplus l_2$. It acts by *inserting* the segment $\{k\}$into this cut: $\text{DEMAND}(T,\ x) \leftarrow l_1 \oplus \{k\} \oplus l_2$;
- INSERT-with-RELOAD: its parameters are some demand node k' in K', a cut $c = (l_1,\ l_2)$ of the sequence $\text{RELOAD}(T,\ k')$, and a non active reload node x. It acts by:
  - inserting the segment $\{x\}$ into the cut c: $\text{RELOAD}(T,\ x) \leftarrow l_1 \oplus \{x\} \oplus l_2$;
  - making x be active and setting: DEMAND $(T,\ x) \leftarrow \{k\}$and $\text{RELOAD}(T,\ k) \leftarrow Nil$

Then we can propose a first insertion greedy algorithm for dealing with **APSCP**:

APSCP-INSERTION **algorithm**

Randomly define a linear ordering ρ on the elements of K;
Initialize T to a tree reduced to the root node *Depot*;
For $k \in K$ **do** */* K being scanned according to the linear order* ρ **/*
  Choose an insertion operator I and the related parameter u, such that the insertion of k through $I(u)$ induces the smallest possible increase of *Tree-Cost*(T);
  Apply $I(u)$ to T;

Remark 4.1. In apscp-insertion algorithm the insertion operator I is chosen among INSERT-SIMPLE and INSERT-with-RELOAD. The related parameter u is chosen in such a way:

- $u = (x, (l_1, l_2))$ in case $I =$ INSERT-SIMPLE;
- $u = (k', (l_1, l_2), x)$ in case $I =$ INSERT-with-RELOAD.

Remark 4.2. In order to save time, we filter the search for the good value of the parameter u. The filtering process is performed by using, for any node x, y in X_R, small sets $N(x)$, $N(y)$ of neighbours of x and y, together with a middle z of x and y, and by imposing conditions related to those objects when dealing with the various components of u.

Of course, the algorithm APSCP-INSERTION may be embedded into the following *Monte-Carlo* Scheme:

Monte-Carlo scheme for APSCP-INSERTION

Parameter Δ:
For i = 1 to Δ
  Run the APSCP-INSERTION Procedure;
Keep the best result.

4.2. Descent algorithms based on variable neighborhood search (VNS)

We perform a variable neighborhood search (VNS) in order to improve solutions obtained by the insertion algorithm APSCP-INSERTION. This VNS process involves six local transformation operators which act through side effect on some bipartite ordered tree T consistent with X and K, and they modify T:

- MOVE-RELOAD: its parameters are some active reload node x and some non active reload node y. It replaces x by y in T;
- MOVE-RELOADS: its parameters are two different demand nodes k and k', a segment l of RELOAD(T, k) and a cut $c = (l_1, l_2)$ of RELOAD(T, k'). It removes l from RELOAD(T, k) and it inserts it into the cut c. Its precondition is that k does not *dominate* k' in the tree T, *i.e.*, that k cannot be obtained from k' through a succession of applications of the FATHER operator;

- MOVE-RELOADS1: its parameters are some demand node k, some segment l of RELOAD(T, k) which induces a decomposition RELOAD(T, k) $= l_3 \oplus l \oplus l_4$, and a cut $c = (l_1, l_2)$ of $l_3 \oplus l_4$. It first removes l from RELOAD(T, k) and next insert it into the cut c: RELOAD(T, k) $\leftarrow l_1 \oplus l \oplus l_2$;
- MOVE-DEMANDS: its parameters are two different active reload nodes x and y, a segment l of DEMAND(T, x), and a cut $c = (l_1, l_2)$ of DEMAND(T, x'). It removes l from DEMAND(T, x) and it inserts it into the cut c. In case DEMAND(T, x) $= l$, it removes the reload node x from T, which becomes non active. Its precondition is that x does not *dominate* y in the tree T;
- MOVE-DEMAND1: its parameters are a reload node x, a segment l of DEMAND(T, x) which induces a decomposition DEMAND(T, x) $= l_3 \oplus l \oplus l_4$, and a cut $c = (l_1, l_2)$ of $l_3 \oplus l_4$. It first removes l and next inserts it into the cut c: DEMAND(T, x) $\leftarrow l_1 \oplus l \oplus l_2$;
- MOVE-DEMANDS-RELOAD: it takes an active reload node x, a non active reload node y, a demand node k, a segment l of DEMAND(T, x) and a cut $c = (l_1, l_2)$ of RELOAD(T, k). It first turns y into an active reload node, next removes l from DEMAND(T, x), inserts it into DEMAND(T, y), and inserts the segment $\{y\}$ into c. In case l = DEMAND(T, x), it turns x into a non active reload node. Its precondition is that k is *dominated* by no demand node k' in l.

APSCP-INSERTION will initialize the following APSCP-DESCENT descent algorithm:

Algorithm APSCP-DESCENT

Initialize the tree T through APSCP-INSERTION;
Initialize the filtering threshold value$H$;
Stop $\leftarrow$ **false**;
While *Stop* $=$ ***false*** **do**
  Search (in a filtered way) parameter values u for some operator I in the above mentioned local transformation operators, in such a way that applying I to T and u improves Tree-Cost(T);
  If the search fails **then** $H \leftarrow H/2$;
  If H *is small enough* **then** *Stop*$\leftarrow$ **true**;

Remark 4.3. In algorithm APSCP-DESCENT, the operators are tried in the same order as they were listed above. As for the way parameters are tried, we use the same kind of filtering device as for the case of the insertion operators. The threshold parameter H is involved in this filtering process, in such a way that the smallest H is, the largest is the domain of the parameter values which are effectively tried.

Additional examples and illustrations of this part are available in [35].

5. Experiments

We have been performing experiments, on a PC equipped with an Intel Xeon 1.86 GHz CPU and 3.25 Go Ram, while using a Visual Studio C++ compiler, and while focusing on several points:

- the ability of APSCP-INSERTION and APSCP-DESCENT to get in a fast way solutions close to the optimal solutions;
- the characteristics of the solutions: number of reload nodes involved in the solution, impact of preemption.

In order to do this, we performed several tests, while using node sets X and distance matrices D proposed by the TSPLIB libraries, and by selecting origin/destination pairs $(o_k,\ d_k,\ k \in K)$ in a random way inside the set X. We dealt with instances which involves from 20 to 300 nodes, and from 10 to 100 origin destination pairs, and, in case of small instances, we got exact results through the use of the ILP formulation of Section 3.

5.1. Experiments about the APSCP-INSERTION algorithm

Our first experiment consists in running the APSCP-INSERTION Monte-Carlo scheme with $\Delta = 100$ and keeping track, for every instance, of the following quantities:

- REF: optimal *Tree-cost* value;
- MIN (MAX, AVG) minimal (maximal, average) *Tree-cost* value obtained through Δ iterations of APSCP-INSERTION;
- EGI: gap (in %) between REF and the solution produced by the APSCP-INSERTION Monte Carlo scheme (MIN value);
- REL: mean number of active reload nodes involved in a solution produced by APSCP-INSERTION;
- DEM/REL: mean number of demands related to every reload node x (length of the list DEMAND(x)), for x different from *Depot*;
- CPU: CPU mean time (in milliseconds) for any iteration of APSCP-INSERTION

The results which we get may be summarized in Tables 1 to 4:

Comments: The instances related to Table 4 are specific in the sense that they have been obtained through a particular construction of the graph: reload nodes are always located in the path between an origin and its destination, so there is no extra cost to go to a reload node. The consequence is that we know in advance their optimal value which may be computed as the sum $\sum_{k \in K} D_{o_k d_k}$.

Globally, Tables 1–4 show that APSCP-INSERTION allows us to get in a fast way solutions which are not too far from the optimal ones. Still, we notice that our greedy scheme is in trouble when it is supposed to create reload nodes.

TABLE 1. Tests performed on 10 instances which we got from the gr24 instance with 49 nodes of the TSPLIB library by randomly sorting 12 demands, and no reload nodes which is not the depot node, the copy of an origin node o_k or the copy of a destination node d_k, $k \in 1 \dots 12$.

Instance	REF	MIN	MAX	AVG	EGI	REL	DEM/REL	CPU
Gr24_v01	24 654	25 023	27 883	26 391	1.5	0.06	2.83	<15
Gr24_v02	21 395	21 424	23 654	22 371	0.136	0.43	1.60	<15
Gr24_v03	22 834	23 363	26 825	24 760	2.32	0.36	2.51	<15
Gr24_v04	23 255	23 444	25 513	24 140	0.813	0.41	2.47	<15
Gr24_v04	23 993	23 993	27 373	25 261	0	0.31	2.03	<15
Gr24_v06	23 233	23 233	25 584	24 427	0	0.54	1.71	<15
Gr24_v07	20 224	20 283	22 594	20 906	0.292	0.11	1.72	<15
Gr24_v08	20 865	21 124	23 334	21 873	1.24	0.61	1.13	15
Gr24_v09	23 054	23 073	25 684	24 014	0.0824	0.43	3	<15
Gr24_v10	26 704	26 963	29 843	28 303	0.97	0.16	4.4	15

TABLE 2. Tests performed on 10 instances which we got from the hk48 instance with 97 nodes of the TSPLIB library by randomly sorting 24 demands, and no reload nodes which is not the depot node, the copy of an origin node o_k or the copy of a destination node d_k, $k \in 1 \dots 24$.

Instance	REF	MIN	MAX	AVG	EGI	REL	DEM/REL	CPU
Hk48_v01	358 048	375 447	405 299	387 605	4.86	0.97	1.67	<15
Hk48_v02	280 579	291 800	325 548	306 673	4	1.44	1.39	<15
Hk48_v03	318 959	325 027	351 678	338 017	1.9	0.95	3.25	<15
Hk48_v04	315 118	322 908	347 647	336 123	2.47	0.53	2.46	15
Hk48_v04	320 578	327 709	360 167	340 639	2.22	0.27	2.80	15
Hk48_v06	306 000	318 838	351 007	334 372	4.2	0.59	4.42	<15
Hk48_v07	314 127	330 528	370 107	347 588	5.22	0.52	3.04	16
Hk48_v08	342 390	350 960	381 719	366 652	2.5	0.53	4.57	16
Hk48_v09	337 327	343 127	370 387	355 734	1.72	0.54	3.24	15
Hk48_v10	330 119	339 509	373 190	355 208	2.84	0.73	1.61	16

5.2. EXPERIMENTS ABOUT THE APSCP-DESCENT

Our second experiment consists in running APSCP-DESCENT from a solution provided by only one application of APSCP-INSERTION, and keeping track, for any instance, of the quantities:

- REF: optimal value;
- VAL: the cost value obtained through APSCP-DESCENT;

TABLE 3. Tests performed on 10 instances which we got from the gr120 instance with 241 nodes of the TSPLIB library by randomly sorting 60 demands, and no reload nodes which is not the depot node, the copy of an origin node o_k or the copy of a destination node d_k, $k \in 1 \dots 60$.

Instance	REF	MIN	MAX	AVG	EGI	REL	DEM/REL	CPU
Gr_120_v01	Unknown	327070	344139	335480	*	0.42	5.13	78
Gr_120_v02	Unknown	332 679	353 240	343 622	*	0.64	4.60	78
Gr_120_v03	Unknown	326 189	345 859	335 567	*	0.27	8.95	78
Gr_120_v04	Unknown	366 750	380 409	374 350	*	0.36	5.74	78
Gr_120_v05	Unknown	310 179	330 002	320 326	*	1.53	3.05	78
Gr_120_v06	Unknown	319 619	337 921	327 146	*	0.64	3.24	78
Gr_120_v07	Unknown	285 989	302 602	293 394	*	0.70	4.46	78
Gr_120_v08	Unknown	345 100	365 379	355 230	*	0.56	7.33	78
Gr_120_v09	Unknown	333 909	350 171	342 549	*	0.47	11.11	78
Gr_120_v10	318 099	337 259	354 809	345 393	6.02	0.36	9.19	78

TABLE 4. Tests performed on 10 instances called *RELm* , which we built in such a way that the instance contains m demands, $p = 2m+1$ reload nodes and $n= (2m + p+1)$ nodes; its optimal value is the sum $\sum_{k \in K} D_{o_k d_k}$ and that the related optimal solution involves all the reload nodes.

Instance	REF	MIN	MAX	AVG	EGI	REL	DEM/REL	CPU
REL15	11 843	13 045	17 088	14 665	10.1	0.11	0.68	<15
REL27	17 464	19 554	23 322	21 841	12.0	0.35	2.40	16
REL39	21 053	24 235	28 386	26 194	15.1	0.54	2.92	31
REL45	22 323	26 321	29 924	27 658	17.9	0.54	2.85	47
REL57	24 142	28 117	31 813	30 000	16.5	0.91	4.50	78
REL81	26 036	30 734	35 302	32 966	18.0	1.76	5.61	141
REL93	26 494	32 077	36 486	34 092	21.1	2.87	4.83	187
REL105	26 775	32 830	36 434	34 655	22.6	3.39	4.83	250
REL128	27 042	33 773	38 796	35 654	24.9	5.92	4.70	375
REL141	27 098	33 664	37 496	36 035	24.2	6.86	5.17	453

- EGI: gap (in %) between REF and the initial solution produced by APSCP-INSERTION;
- ED: gap between REF and the final solution produced by APSCP-DESCENT;
- REL: number of reloads involved in the solution produced by APSCP-DESCENT;

TABLE 5. Tests performed on 10 instances which we got from the gr24 instance with 49 nodes of the TSPLIB library by randomly sorting 12 demands, and no reload nodes which is not the depot node, the copy of an origin node o_k or the copy of a destination node d_k, $k \in 1 \dots 12$.

Instance	REF	VAL	EGI	ED	ETDS	REL	TNB	CPU (ms)
Gr24_v01	24 654	24 654	10.3	0	10.3	1	12	93
Gr24_v02	21 395	21 914	5.0	2.4	2.6	1	3	47
Gr24_v03	22 834	22 874	9.9	0.1	9.8	1	9	46
Gr24_v04	23 255	23 435	3.9	0.7	3.2	2	4	31
Gr24_v04	23 993	23 993	0	0	0	0	0	31
Gr24_v06	23 233	23 763	8.3	2.2	6.1	0	3	31
Gr24_v07	20 224	20 244	1.2	0.1	0.2	1	3	78
Gr24_v08	20 865	21 084	4.9	1.0	3.9	1	6	31
Gr24_v09	23 054	23 054	4.9	0	4.9	1	7	47
Gr24_v10	26 704	26 754	4.9	0.2	3.7	1	7	63

- ETDS: part of the gap between EGI and ED which is induced by the operators MOVE-DEMANDS, MOVE-DEMANDS1 and MOVE-DEMAND-with-RELOAD;
- REL: number of active reloads involved in the solution produced by APSCP-DESCENT;
- TNB: number of times a local transformation operator is effectively applied inside the APSCP-DESCENT process;
- CPU: Cpu running time.

The results which we got may be summarized in Table 5 to 8:

Comments: Though APSCP-DESCENT does not involve any of the classical control mechanisms which allow dealing with local optima (simulated annealing, tabu search...), we see that the operators which derive from our tree representation enable us to get very satisfactory results in a fast way.

5.3. EXPERIMENTS ABOUT THE IMPACT OF THE NO PREEMPTION HYPOTHESIS

We have just been dealing with a preemptive version of a routing problem which is usually handled in a non preemptive way. An interesting question which arises in a natural way is about the evaluation of the true impact of this no preemption hypothesis.

In order to estimate the gap between the optimal values which are respectively related to preemptive and no-preemptive version of the ASCP, we perform a third experiment which consists in running APSCP-DESCENT, from a solution which is provided by only one application of APSCP-INSERTION while forbidding in both cases the creation of reloads (we call the related restricted processes "no preemptive

TABLE 6. Tests performed on 10 instances which we got from the hk48 instance with 97 nodes of the TSPLIB library by randomly sorting 24 demands, and no reload nodes which is not the depot node, the copy of an origin node o_k or the copy of a destination node d_k, $k \in 1 \dots 24$.

Instance	REF	VAL	EGI	ED	ETDS	REL	TNB	CPU (ms)
Hk48_v01	358 048	358 349	5.2	0.1	5.1	2	18	8
Hk48_v02	280 579	284 411	7.9	1.3	6.3	4	20	11
Hk48_v03	318 959	322 879	5.7	1.2	4.3	2	12	200
Hk48_v04	315 118	316 362	5.1	0.4	4.6	5	24	7
Hk48_v04	320 578	323 508	7.1	0.9	6.1	1	23	300
Hk48_v06	306 000	310 963	12.6	1.6	10.4	5	38	14
Hk48_v07	314 127	319 009	12.9	1.5	11.1	2	28	12
Hk48_v08	342 390	342 551	6.7	0.05	6.6	4	26	18
Hk48_v09	337 327	338 500	6.4	0.3	6.0	3	35	9
Hk48_v10	330 119	334 881	6.4	1.4	4.8	4	19	7

TABLE 7. Tests performed on 10 instances which we got from the gr120 instance with 241 nodes of the TSPLIB library by randomly sorting 60 demands, and no reload nodes which is not the depot node, the copy of an origin node o_k or the copy of a destination node d_k $k \in 1 \dots 60$.

Instance	REF	VAL	EGI	ED	ETDS	REL	TNB	CPU (s)
Gr_120_v01	Unknown	313 315	*	*	5.2	6	89	130
Gr_120_v02	Unknown	318 870	*	*	8.9	1	116	189
Gr_120_v03	Unknown	314 493	*	*	6.2	4	96	125
Gr_120_v04	Unknown	360 902	*	*	3.5	3	107	145
Gr_120_v05	Unknown	293613	*	*	9.8	4	99	125
Gr_120_v06	Unknown	303 441	*	*	7.7	2	90	80
Gr_120_v07	Unknown	265 713	*	*	6.4	4	84	86
Gr_120_v08	Unknown	330 823	*	*	6.9	4	77	94
Gr_120_v09	Unknown	314 073	*	*	8.5	4	105	70
Gr_120_v10	318 099	318 913	9.5	0.25	9.1	4	96	103

APSCP-DESCENT" and "no preemptive APSCP-INSERTION"). Every time this composite process is performed we keep memory of the quantities:

- preemptive ED: gap between optimal preemptive value and the preemptive solution produced by preemptive APSCP-DESCENT;
- no preemptive ED: gap between optimal preemptive value and the no-preemptive solution produced by no-preemptive APSCP-DESCENT;
- preemptive CPU: Cpu running time for preemptive APSCP-DESCENT

TABLE 8. Tests performed on 10 instances *RELm* , which we built in such a way that the instance contains m demands, $p = 2m+1$ reload nodes and $n = (2m + p + 1)$ nodes; its optimal value is the sum $\sum_{k \in K} D_{o_k d_k}$, and that the related optimal solution involves all the reload nodes.

Instance	REF	VAL	EGI	ED	ETDS	REL	TNB	CPU (s)
REL15	11 843	11 953	16.7	0.9	15.8	4	10	0.04
REL27	17 464	17 583	27	0.7	36.3	8	28	0.34
REL39	21 053	21 053	29.9	0	29.9	12	56	1.5
REL45	22 323	22 605	20.8	1.2	19.6	14	46	2.9
REL57	24 142	24 225	28.4	0.34	28.4	18	78	8.4
REL81	26 036	26 279	19.3	0.93	18.3	26	79	25
REL93	26 494	26 626	26.6	0.49	26.2	30	112	39
REL105	26 775	26 851	33.1	0.28	30.1	34	134	78
REL128	27 042	27 067	37	0.09	36.9	42	162	232
REL141	27 098	27 211	33.1	0.4	32.4	46	202	328

TABLE 9. Tests performed on the 10 instances *RELm*.

Instance	Preemptive ED	No preemptive ED	Preemptive CPU (s)	No preemptive CPU (ms)
REL15	0.90	13.7	0.04	15
REL27	0.70	27.1	0.34	15
REL39	0.00	26.9	1.50	15
REL45	1.20	13.9	2.90	15
REL57	0.34	24.8	8.40	16
REL81	0.93	17.2	25	31
REL93	0.49	23.3	39	31
REL105	0.28	31.3	78	31
REL128	0.09	35.3	232	31
REL141	0.40	37.4	328	47

- no-preemptive CPU: Cpu running time for no-preemptive APSCP-DES CENT

The results which we get may be summarized in Table 9 and 10:

Comments: Through these results we see that, in some case, allowing preemption may induce a great improvement of solutions. However, handling solutions in a preemptive way is much more time consuming.

TABLE 10. Tests performed on the 10 instances which we got from the hk48 instance with 97 nodes.

Instance	Preemptive ED	No preemptive ED	Preemptive CPU (ms)	No preemptive CPU (ms)
Hk48_v01	0.1	7.64	80	<15
Hk48_v02	1.3	7.08	110	<15
Hk48_v03	1.2	2.92	200	<15
Hk48_v04	0.4	5.34	70	<15
Hk48_v04	0.9	4.59	300	<15
Hk48_v06	1.6	2.55	140	<15
Hk48_v07	1.5	10.5	120	<15
Hk48_v08	0.05	5.52	180	<15
Hk48_v09	0.3	8.05	90	<15
Hk48_v10	1.4	5.33	70	<15

6. Conclusion

We have been dealing here with a preemptive demand routing problem with capacity constraints, and we showed how it was possible to turn it into a non constrained tree construction problem in such a way that we could solve it in an efficient way through simple greedy and descent processes. It would be interesting to study to what extent our approach could be suitable to the handling of more general routing and scheduling problems.

References

[1] S. Anily and R. Hassin, The swapping problem. *Networks* **22** (1992) 419–433.
[2] S. Anily, M. Gendreau and G. Laporte, The preemptive swapping problem on a tree. Submitted to *Networks* (2009).
[3] N. Ascheuer, L. Escudero, M. Grötschel and M. Stoer, A cutting plane approach to the sequential ordering problem (with applications to job scheduling in manufacturing). *SIAM J. Optim.* **3** (1993) 25–42.
[4] N. Ascheuer, M. Jünger and G. Reinelt, A Branch and Cut algorithm for the Asymmetric Traveling Salesman Problem with Precedence Constraints. *Comput. Optim. Appl.* **17** (2000) 61–84.
[5] M.J. Atallah and S.R. Kosaraju, Efficient solutions to some transportation problems with applications to minimizing robot arm travel. *SIAM J. Comput.* **17** (1988) 849.
[6] E. Balas, M. Fischetti and W. Pulleyblank, The precedence constrained asymmetric traveling salesman problem. *Math. Program.* **68** (1995) 241–265.
[7] G. Berbeglia, J.F. Cordeau, I. Gribkovskaia and G. Laporte, Static pickup and delivery problems: a classification scheme and survey. *TOP: An Official Journal of the Spanish Society of Statistics and Operations Research* **15** (2007) 1–31.
[8] C. Bordenave, M. Gendreau and G. Laporte, *A branch-and-cut algorithm for the preemptive swapping problem.* Technical Report CIRRELT-2008-23 (2008).
[9] C. Bordenave, M. Gendreau and G. Laporte, Heuristics for the mixed swapping problem. *Comput. Oper. Res.* **37** (2010) 108–114.

[10] S. Chen and S. Smith, *Commonality and genetic algorithms*. Technical Report CMU-RI-TR-96-27, Robotics Institute, Carnegie Mellon University, Pittsburgh, PA (1996).
[11] J.F. Cordeau, M. Iori, G. Laporte and J.J. Salazar-González, A branch-and-cut algorithm for the pickup and delivery traveling salesman problem with LIFO loading. *Networks* **55** (2010) 46–59.
[12] C.E. Cortés, M Matamala and C. Contardo, The Pickup and Delivery Problem with Transfers: Formulation and Solution Approaches, in *VII French – Latin American Congress on Applied Mathematics*. Springer (2005).
[13] I. Dumitrescu, *Polyhedral results for the pickup and delivery travelling salesman problem*. Technical Report, CRT-2005-27 (2005).
[14] M.T. Fiala Timlin, *Precedence constrained routing and helicopter scheduling*. M. Sc. thesis, Department of Combinatorics and Optimization University of Waterloo (1989).
[15] M.T. Fiala Timlin and W.R. Pulleyblank, Precedence constrained routing and helicopter scheduling: heuristic design. *Interfaces* **22** (1992) 100–111.
[16] G.N. Frederickson and D.J. Guan, Preemptive ensemble motion planning on a tree. *SIAM J. Comput.* **21** (1992) 1130.
[17] G.N. Frederickson and D.J. Guan, Nonpreemptive ensemble motion planning on a tree. *J. Algorithms* **15** (1993) 29–60.
[18] G.N. Frederickson, M.S. Hecht and C.E. Kim, Approximation algorithms for some routing problems. *SIAM J. Comput.* **7** (1978) 178.
[19] L.M. Gambardella and M. Dorigo, An ant colony system hybridized with a new local search for the sequential ordering problem. *INFORMS J. Comput.* **12** (2000) 237–255.
[20] L. Gouveia and P. Pesneau, On extended formulations for the precedence constrained asymmetric traveling salesman problem. *Networks* **48** (2006) 77–89.
[21] H. Hernández-Pérez and J. Salazar-González, The multicommodity one-to-one pickup-and-delivery traveling salesman problem. *Eur. J. Oper. Res.* **196** (2009) 987–995.
[22] B. Kalantari, A.V. Hill and S.R. Arora, An algorithm for the traveling salesman problem with pickup and delivery customers. *Eur. J. Oper. Res.* **22** (1985) 377–386.
[23] M. Lacroix, *Le problème de ramassage et livraison préemptif : complexité, modèles et polyèdres*. Ph.D. thesis, Université Blaise Pascal, Clermont-Ferrand II (2009).
[24] S. Lin, Computer solutions to the traveling salesman problem. *Bell System Technical Journal* **44** (1965) 2245–2269.
[25] J. Little, K. Murty, D. Sweeney and C. Karel, An algorithm for the traveling salesman problem. *Oper. Res.* **11** (1963) 972–989.
[26] S. Mitrovic-Minic and G. Laporte, The pickup and delivery problem with time windows and transshipment. *INFOR* **44** (2006) 217–227.
[27] R. Montemanni, D.H. Smith and L.M. Gambardella, A heuristic manipulation technique for the sequential ordering problem. *Comput. Oper. Res.* **35** (2008) 3931–3944.
[28] P. Oertel, *Routing with Reloads*. Doktorarbeit, Universität zu Köln (2000).
[29] S.N. Parragh, K.F. Doerner and R.F. Hartl, A survey on pickup and delivery problems: Part II: Transportation between pickups and delivery locations. *Journal für Betriebswirtschaft* **58** (2008) 21–51.
[30] H.N. Psaraftis, A Dynamic Programming solution to the single-vehicle many to-many immediate request dial-a-ride problem. *Transp. Sci.* **14** (1980) 130–154.
[31] H.N. Psaraftis, k-interchange procedures for local search in a precedence constrained routing problem. *Eur. J. Oper. Res.* **13** (1983) 391–402.
[32] J. Renaud, F. Boctor and J. Ouenniche, A heuristic for the pickup and delivery traveling salesman problem. *Comput. Oper. Res.* **27** (2000) 905–916.
[33] J. Renaud, F. Boctor and G. Laporte, Perturbation heuristics for the pickup and delivery traveling salesman problem. *Comput. Oper. Res.* **29** (2002) 1129–1141.
[34] K.M. Ruland and E.Y. Rodin, The pickup and delivery problem: Faces and branch-and-cut algorithm. *Comput. Math. Appl.* **33** (1997) 1–13.
[35] H. Toussaint, *Algorithmique rapide pour les problèmes de tournées et d'ordonnancement*. Ph.D. thesis, Université Blaise Pascal, Clermont-Ferrand II (2010).