Available online at www.sciencedirect.com
ScienceDirect
ELSEVIER
Electronic Notes in Discrete Mathematics 36 (2010) 41–48
Electronic Notes in DISCRETE MATHEMATICS
www.elsevier.com/locate/endm

Tree based heuristics for the preemptive asymmetric stacker crane problem

A. Quilliot [1] M. Lacroix [1] H. Toussaint [1] H. Kerivin [2]

[1] *LIMOS, CNRS UMR 6158, Université Blaise-Pascal, Clermont-Ferrand II, France*

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

Abstract

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

Keywords: preemptive stacker crane problem, reloads, routing, local search, heuristic design.

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

[1] Email: {quilliot,lacroix,toussain}@isima.fr
[2] Email: kerivin@clemson.edu

1571-0653/$ – see front matter © 2010 Elsevier B.V. All rights reserved.
doi:10.1016/j.endm.2010.05.006

using a given set of vehicles, have been intensively studied for decades. Many variants have been considered and one can refer to [2],[6] for surveys on these problems and methods. The Stacker Crane Problem (SCP) is a pickup and delivery problem which involves only one vehicle, which can deal with only one demand unit at the same time.

A rough description of the Stacker Crane Problem (SCP) comes as follows: G being some transit network whose oriented links or arcs are endowed with lengths or costs and which is provided with some specific *Depot* node, we are required to schedule the route of a single vehicle V, which is supposed 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 route Γ inside the network G, which is going to start and end into *Depot* while making possible for V to handle every demand $k \in K$. Solving the SCP will mean computing this route in such a way that it is the shortest possible. In the Preemptive Stacker Crane Problem (PSCP), any load unit related to demand k 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 of demand k reaches the destination node d_k. In case the cost function, which to any arc (x, y), makes correspond some cost $DIST(x, y)$, is asymmetric, we talk about Asymmetric SCP.

The SCP was first introduced by Frederickson et al. in [5], under its non preemptive symmetric form. These authors proved its NP-hardness by using a reduction from the TSP. They also got a 9/5-approximation scheme for this problem. Atallah and Kosaraju [1] were the first to consider the preemptive version of the symmetric SCP. They studied both non-preemptive and preemptive versions of the symmetric SCP in the case 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 [3,4] studied both preemptive and non-preemptive versions of the symmetric SCP in the case when the underlying graph is a tree.

The focus of this paper will be on the preemptive asymmetric SCP, which we shall denote by APSCP. We are first going to set our problem in a formal way. Next we shall state some structural results which will allow us to turn the problem into a non constrained tree design problem. This reformulation of the problem will lead us to design in a natural way local search heuristic scheme together with a linear integer programming model, which will be implemented

and tested in section 5, providing us with satisfactory numerical results.

2 Formal Description and Structural Results

2.1 Notations about sets and lists

For any sequence $\Gamma=\{x_1,\ldots,x_n\}$ we denote by $Rank(\Gamma,x)$ the rank i of $x=x_i$ in Γ. The first (last) element of Γ is denoted by $First(\Gamma)$ ($Last(\Gamma)$). We denote by $\oplus$ the *concatenation* operator, which concatenates two sequences $\Gamma=\{x_1,\ldots,x_n\}$ and $\Gamma'=\{y_1,\ldots,y_n\}$ into a unique sequence $\Gamma\oplus\Gamma'=\{x_1,\ldots,x_n,y_1,\ldots,y_n\}$. A *segment* of Γ is a subsequence $\{x,\ldots,y\}$ of Γ such that $Rank(\Gamma,x)\leq Rank(\Gamma,y)$, and a *cut* of Γ is any decomposition $\Gamma=\Gamma'\oplus\Gamma''$.

2.2 Modelling the Asymmetric Pre-emptive Stacker Crane Problem (APSCP)

In order to get a formal model of the APSCP 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 eventual 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 eventual 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 $X\times X$ indexed distance matrix $DIST$, such that if $x\in Depot\cup X_O\cup X_D$ and if x' is the copy of $x\in X_R$, then $DIST(x,x')=0$. Given an APSCP instance (X,K) defined this way, we easily deduce a notion of valid tour related to this instance. Thus, solving the APSCP instance (X,K) means finding such a valid tour Γ with minimal length $Length(\Gamma)$.

2.3 A Tree Representation of the APSCP Problem

As a matter of fact, some tours with special properties, can be represented as some kind of trees, called a *bipartite ordered trees*. A tree T is a *bipartite ordered tree* if: (1) its nodes can be split into two classes A and B in such a way that nodes in class A have their sons in class B and conversely; (2) 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. We say that a bipartite ordered tree T is *consistent* with the APSCP instance defined by the demand set K and by the node set X ($(X,K)-consistent$) if:

- 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 demand node $k \in K$ appears in T, while only some nodes of $Depot \cup X_R$ appear in T (they define the *active* reload node set $Active(T)$ of T);
- The root of T is the *Depot* node and the terminal nodes (leafs) of T are all demand nodes;
- For any demand node k, its linearly ordered son set $Reload(T,k)$ (which may be empty) is in $Active(T)$ and its father $Father(T,k)$ is in $Active(T)$;
- 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.

For any such a bipartite ordered tree T, we define a cost value $Tree\text{-}Cost(T)$ as follows:

- For any demand node $k \in K$, we set:
  If k is not a terminal node then
  $$Cost\text{-}Demand(T,k) = DIST(o_k, First(Reload(T,k))) + DIST(Last(Reload(T,k)), d_k) + \sum_{\substack{x \in Reload(T,k),\\ x \neq Last(Reload(T,k))}} DIST(x, Succ(Reload(T,k),x))$$
  else $Cost\text{-}Demand(T,k) = DIST(o_k, d_k)$
- For any reload node $x \in Depot \cup X_R$, we set:
  $$Cost\text{-}Reload(T,x) = DIST(x, o_{First(Demand(T,x))}) + DIST(d_{Last(Demand(T,x))},\ x) + \sum_{\substack{k \in Demand(T,x),\\ k \neq Last(Demand(T,x))}} DIST(d_k, o_{Succ(Demand(T,x),\ k)});$$
- $Tree\text{-}Cost(T) = \sum_{k \in K} Cost - Demand(T,k) + \sum_{x \in ACTIVE(T)} Cost - Reload(T,x)$.

We notice that any consistent bipartite tree T can be turned into a valid tour $Tour(T)$, as in figure 1, in such a way that $Length(Tour(T))$=$Tree\text{-}Cost(T)$. We denote by $Tour\text{-}Tree$, the set of all valid tours which may obtained through this process. So our main statement comes as follows:

Theorem 2.1 *Given the APSCP instance (X, K), there exists some optimal solution of (X, K) which is in $Tour\text{-}Tree$. Thus, handling (X, K) can be done by searching for a consistent bipartite ordered tree T, such that $Tree\text{-}Cost(T)$ is the smallest possible.*

Principle of the proof: we start from some optimal tour Γ, and we turn it into a tour in $Tour\text{-}Tree$ while keeping $Length(\Gamma)$ from increasing.

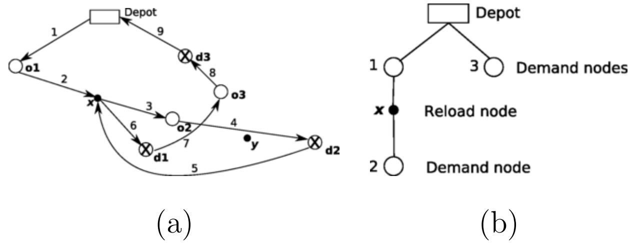

Fig. 1. (a) A tour , (b) The bipartite tree which derives from the tour (a)

3 Tree Based Heuristics for the APSCP Problem

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

Insertion Operators: they 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 $Demand(T, x) = l_1 \oplus l_2$. It acts by inserting the segment $\{k\}$ into this cut: $Demand(T, x) \leftarrow l_1 \oplus k \oplus l_2$.
- INSERT-with-RELOAD: its parameters are some demand node k' in K', a cut $c = (l_1, l_2)$ of the sequence $Reload(T, k')$, and a non active reload node x. It acts by: (1) inserting the segment $\{x\}$ into the cut c: $Reload(T, x) \leftarrow l_1 \oplus \{x\} \oplus l_2$; (2) making x be active and setting: $Demand(T, x) \leftarrow \{k\}$; $Reload(T, k) \leftarrow Nil$.

Local Transformation Operators: they act through side effect on some bipartite ordered tree T consistent with X and K, and they modify T. We use 6 operators (We note $x \leftarrow v$ when the variable x is provided with the value v):

- 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-RELOADSl: 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 $Demand(T,y)$. 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-DEMANDl: 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. Node k must be dominated by no demand node k' in l.

We propose a first insertion greedy algorithm for dealing with APSCP called `APSCP-insert`: we first randomly define a linear ordering ϱ on the elements of K and initialize T to the root node $Depot$. Then, for each $k \in K$, (K being scanned according to the linear order ϱ) we choose the insertion operator I (among INSERT-SIMPLE and INSERT-with-RELOAD) and its related parameter u such that the insertion of k through $I(u)$ induces the smallest possible increase of $Tree\text{-}Cost(T)$. We filter the search for the good value of the parameter u 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 $z = MID(x,y)$ 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-insert` may be embedded into a *Monte-Carlo* Scheme: given a parameter Δ we run the `APSCP-insert` procedure Δ times and keep the best result.

We also propose a descent algorithm `APSCP-desc`: given a tree T initialized with `APSCP-insert`, we 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 $Tree\text{-}Cost(T)$. We use the same kind of filtering device as for the case of the insertion operators.

4 Experiments

We performed two experiments, on PC IntelXeonwith 1.86 GHz, 3.25 Go Ram, while using a Visual Studio C++ compiler. We focused on the accuracy and speed of our algorithms and on the characteristics of the resulting solutions: number of reload nodes involved in the solution, impact of pre-emption. In order to do this, we performed several tests, while using node sets X and distance matrices $DIST$ proposed by the TSPLIB libraries, and by selecting origin/destination pairs in a random way inside the set X. We dealt with instances which involve from 20 to 300 nodes and from 10 to 100 origin destination pairs, and, in case of small instances, got exact results through the use of a LIP formulation, augmented with cutting plane techniques ([7]).

The first experiment consisted in running the `APSCP-insert` Monte-Carlo scheme with $\Delta = 100$, while keeping memory, for every instance, the following quantities: (1) dem.: number of demands, (2) nod.: number of nodes, (3) gap1: gap (in %) between exact theorical value and the best value produced by the `APSCP-insert` Monte Carlo scheme, (4) rel1: Mean number of active reload nodes involved in a solution produced by `APSCP-insert`; (5) cpu1: CPU Mean Time (in seconds) for any iteration of `APSCP-insert`.

The second experiment consisted in running `APSCP-desc` from a solution provided by only 1 application of `APSCP-insert`, while keeping memory, for every instance, the quantities: (1) gapI: gap (in %) between exact theorical value and the initial value produced by `APSCP-insert`, (2) gapF: gap (in %) between exact theorical value and the final value produced by `APSCP-desc`, (3) rel2: number of reloads involved in the solution produced by `APSCP-desc`, (4) cpu2: CPU Running Time (in seconds) for `APSCP-desc`.

The results we got are summarized in table 1.

dem.	nod.	gap1	rel1	cpu1	gapI	gapF	rel2	cpu2
11	46	1.5	0.06	15.10^{-3}	10.3	0	1	0.1
11	46	0.14	0.43	15.10^{-3}	5.0	2.4	1	0.05
23	94	1.9	0.95	15.10^{-3}	5.7	1.2	2	2.0
23	94	2.47	0.53	15.10^{-3}	5.1	0.4	5	0.7
59	238	6.02	0.36	78.10^{-3}	9.5	0.25	4	103

Table 1
Tests performed on 5 instances

5 Conclusion

We have been dealing here with a pre-emptive 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 extend the approach which we presented here in a specific context, and try to deduce efficient approaches for the handling of pre-emption in more general routing and scheduling problems.

References

[1] M.J. Atallah and S.R. Kosaraju. Efficient Solutions to Some Transportation Problems with Applications to Minimizing Robot Arm Travel. *SIAM Journal on Computing*, 17: 849, 1988.

[2] 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 Soc. of Stat. and Operations Res.*, 15(1): 1–31, July 2007.

[3] G.N. Frederickson and DJ Guan. Preemptive Ensemble Motion Planning on a Tree. *SIAM Journal on Computing*, 21: 1130, 1992.

[4] G.N. Frederickson and D.J.Guan. Nonpreemptive ensemble motion planning on a tree. *Journal of Algorithms*, 15(1): 29–60, 1993.

[5] G.N. Frederickson, M.S. Hecht, and C.E. Kim. Approximation Algorithms for Some Routing Problems. *SIAM Journal on Computing*, 7: 178, 1978.

[6] 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(1):21–51, 2008.

[7] M. Lacroix, le problème de ramassage et livraison préemptif : complexité, modèles et polyèdres, Phd thesis, Université Blaise Pascal, Clermont-Ferrand, France, 2009.