The Eighth International Symposium on Operations Research and Its Applications (ISORA'09)
Zhangjiajie, China, September 20–22, 2009

Routing by Mixed Set Programming

Jianyang Zhou

ENGINEST, 1 allée de l'Alzette – 54500 Vandoeuvre-Lès-Nancy – France
zhou@enginest.com

Abstract This paper presents the solution to a routing problem (Pick-up and Delivery with Time Windows) by Mixed Set Programming which solves constraint satisfaction problems over a mixed domain of reals, integers, Booleans, references and sets [8]. To illustrate the method, a complete constraint program in the NCL language [7, 9] is presented.

Keywords Mixed Set Programming, Natural Constraint Language, Pick-up and Delivery with Time Windows, Vehicle Routing

1 Introduction

Pessimistic conclusions are made in [3,4] about the CP (Constraint Programming) method respectively in solving set partitioning problem (SPP) and vehicle routing problem (VRP) though CP is flexible in handling constraints. Linear solvers seem to be efficient in solving SPP and VRP. However, algorithms by linear relaxation may not be flexible enough in tackling complex side constraints.

Based on the above observation, we take SPP and VRP as benchmark problems to evaluate the performance of an exact solver. This paper argues: 1) Linear solver is not the only type of solver that can tackle SPP and VRP; 2) Traditional CP solver is not the only type of solver that is flexible in modeling Constraint Satisfaction Problems (CSP). This paper shows: 1) In [8], MSP (Mixed Set Programming) solves efficiently several hard SPP instances that are not solved by CP; 2) In this paper, a complete MSP model is presented in NCL [7, 9] to illustrate the solution to a hard routing problem – Pick-up and Delivery with Time Windows.

2 Mixed Set Programming

Mixed Set Programming (MSP) is an algorithmic framework for modeling and solving constraint satisfaction problems [8, 9].

By Set Programming we do not mean the simple use of set notations or set variables in a constraint solving system, but rather rigorous and complete set theoretical formulation and reasoning in a systematic way to solve problems such as Set Partitioning, Job-shop Scheduling and Vehicle Routing.

By Mixed Set Programming, we mean:

- Constraint solving and reasoning over a mixed domain of reals, integers,

Booleans, references, and sets;

- Incorporating and combining a simplified form of first order logic, naïve set reasoning, numerical constraints and OR algorithms in a cooperative way. Here, set theoretical formulation is fundamental in problem modeling and set theoretical reasoning is crucial in problem solving.

3 Routing By Set Sorting

Results in this paper are obtained on a PC with a CPU of 1.83GHz.

3.1 Set Sorting By 2 Tuples of Variables

Sorting is fundamental in computer science. In MSP, logical sorting is critical for scheduling and routing. In [6], the solution to the Job-shop problem using integer sorting with 3 tuples of variables is presented; in [8], the solution to the Job-shop problem using set sorting with 3 tuples of variables is presented. This paper presents the set sorting model with 2 tuples of variables. We adopt the following notation convention: Possibly subscripted by i, j, k, l and numbered by n, we take x for integer variable and B for set variable.

$\forall\, i \in [1, n]$ (

$x_i \in [2, n+1]$,

$B_i \prec B_{x_i}$

) ,

$\forall\, i \neq j \in [1, n]$

$x_i \neq x_j$,

To facilitate the presentation, the model supposes that 1 is the source and n+1 is the sink for the sorting chain. Single vehicle routing is illustrated in Figure 1.

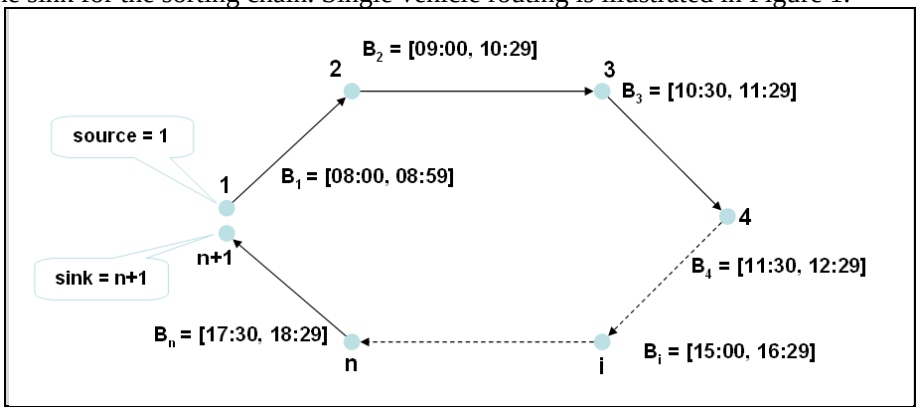

Figure 1: Routing By Set Sorting with 2 Tuples of Variables

3.2 Pick-up & Delivery with Time Windows

In the Pick-up & Delivery Problem with Time Windows (PDPTW) [2], vehicles transport goods from origins to destinations on respecting dynamic load, precedence and coupling constraints in addition to capacity and time window constraints of the vehicle routing problem [5].

In this paper, vehicle routing is viewed as the composed problem of set partitioning and traveling salesman problems. In [8], MSP is showed to be effective in handling set partitioning problem. By combining set partitioning and set sorting models [8], the solving of the pick-up and delivery problem is presented through the following complete NCL program:

```
nbTruck          = O,                          % number of trucks
capacityTruck    = O,                          % truck capacity
nbOrder          = O,                          % number of orders
TRUCK            = [1, nbTruck],
ORDER            = [1, nbOrder],

∀ i ∈ TRUCK (
sourceTruck_i  = – i,                          % origin of truck i
sinkTruck_i    = – (i + nbTruck)               % destination of truck i
),

depot            = –1,                         % one depot
SOURCE = { ∀ i ∈ TRUCK sourceTruck_i },        % origins
SINK             = { ∀ i ∈ TRUCK sinkTruck_i }, % destinations
SOURCEORDER =  ORDER ∪ SOURCE,
ORDERSINK        =  ORDER ∪ SINK,
SOURCEORDERSINK =  ORDER ∪ SOURCE ∪ SINK,

∀ i ∈ SOURCEORDERSINK (                        % explicit declaration
xOrder_i                 :: 0,                 % abscissa
yOrder_i                 :: 0,                 % ordinate
demandOrder_i            :: 0,                 % demand
w1Order_i                :: 0,                 % lower bound of time window
w2Order_i                :: 0,                 % upper bound of time window
serviceTimeOrder_i       :: 0,                 % service time
idxPickupOrder_i         :: 0,                 % index to pickup order
idxDelivrOrder_i         :: 0,                 % index to delivery order
TimeWinOrder_i  = [w1Order_i, w2Order_i],      % time window
loadOrder_i  ∈ [0, capacityTruck],             % dynamic truck load at order i
```

$truckOrder_i \in TRUCK$, % truck for an order
),
$\forall\ i \in \{depot\} \cup ORDER$ (% input for depot & orders
$xOrder_i = O$,
$yOrder_i = O$,
$demandOrder_i = O$,
$w1Order_i = O$,
$w2Order_i = O$,
$serviceTimeOrder_i = O$,
$idxPickupOrder_i = O$,
$idxDelivrOrder_i = O$
),
$\forall\ i \in SOURCE \cup SINK$ (% same origins & destinations
$xOrder_i = xOrder_{depot}$,
$yOrder_i = yOrder_{depot}$,
$demandOrder_i = demandOrder_{depot}$,
$TimeWinOrder_i = TimeWinOrder_{depot}$,
$serviceTimeOrder_i = serviceTimeOrder_{depot}$,
$idxPickupOrder_i = 0$,
$idxDelivrOrder_i = 0$
),

$PICKUPORDER = \{ \forall i \in ORDER \ (demandOrder_i > 0 \ ? \ i : 0) \} \setminus 0$,

$\forall\ i, j \in SOURCEORDERSINK$ (
$dOrderOrder_{i,j} = \lfloor \sqrt{(xOrder_i - xOrder_j)^2 + (yOrder_i - yOrder_j)^2} + 0.5 \rfloor$, % distance matrix
$tOrderOrder_{i,j} = dOrderOrder_{i,j} + serviceTimeOrder_i$ % time matrix
),
$\forall\ i \in TRUCK$ (
$OrderTruck_i \subset ORDER$, % set of orders of truck i
$OrderSourceTruck_i = \{ sourceTruck_i \} \cup OrderTruck_i$,
$OrderSinkTruck_i = \{ sinkTruck_i \} \cup OrderTruck_i$,
$OrderSourceSinkTruck_i = OrderSourceTruck_i \cup \{ sinkTruck_i \}$,
$truckOrder_{sourceTruck_i} = i$,
$truckOrder_{sinkTruck_i} = i$,
$loadOrder_{sourceTruck_i} = 0$,

$\text{loadOrder}_{\text{sinkTruck}_i} = 0$
),

$\text{ORDER} = \cup_{i \in \text{TRUCK}} \text{OrderTruck}_i$, % set partitioning over orders
$\forall\, i < j \in \text{TRUCK}$
$\text{OrderTruck}_i \cap \text{OrderTruck}_j = \varnothing$,

$\forall\, i \in \text{SOURCEORDER}$ (
$\text{TimeOrder}_i = [\text{t1Order}_i, \text{t2Order}_i]$, % work time from i to next
$\text{TimeOrder}_i \subset \text{TimeWinOrder}_{\text{depot}}$,
$\text{nextOrderOrder}_i \in \text{ORDERSINK}$, % successor of order i
$\text{TimeOrder}_i \prec \text{TimeOrder}_{\text{nextOrder}_i}$, % set sorting for routing
$\#\text{TimeOrder}_i = \text{tOrderOrder}_{i,\, \text{nextOrderOrder}_i}$, % distance from order i to next
$i \in \text{OrderSourceTruck}_{\text{truckOrder}_i}$,
$\text{t1Order}_i \in \text{TimeWinOrder}_i$, % time window constraint
$\text{loadOrder}_{\text{nextOrder}_i}$ % dynamic load constraint
$= \text{loadOrder}_i + \text{demandOrder}_{\text{nextOrder}_i}$
),
$\forall\, i \neq j \in \text{SOURCEORDER}$
$\text{nextOrder}_i \neq \text{nextOrder}_j$,
$\forall\, i \in \text{PICKUPORDER}$ (
$\text{TimeOrder}_i \prec \text{TimeOrder}_{\text{idxDelivrOrder}_i}$, % precedence constraint
$\text{truckOrder}_i = \text{truckOrder}_{\text{idxDelivrOrder}_i}$ % coupling constraint
),
$\forall\, i \in \text{TRUCK}$ (
$\text{t1Order}_{\text{sourceTruck}_i} = 0$, % trucks depart always at time 0
$\forall\, j \in \text{SourceOrderTruck}_i$
$\text{nextOrder}_j \in \text{OrderSinkTruck}_i$,
$\text{activeTruck}_i \equiv \text{OrderTruck}_i \neq \varnothing$ % active iff it carries an order
),
$\forall\, i < j \in \text{TRUCK}$ (% symmetry breaking
$\text{activeTruck}_j \rightarrow \text{activeTruck}_i$,
sup $\text{OrderSourceTruck}_i > $ *sup* $\text{OrderSourceTruck}_j$
),
$\forall\, j \in \text{PICKUPORDER}$ (

$\forall$ i $\in$ TRUCK (
dSumOrderTruck$_{i,j}$ =
$\sum_{k \in OrderTruck_i}$ max(dOrderOrder$_{j,k}$, dOrderOrder$_{idxDelivrOrder_{j,k}}$),
dOrderTruck$_{i,j}$ = $\lfloor$dSumOrderTruck$_{i,j}$ / #OrderTruck$_i$ + 0.5$\rfloor$,
), % order-to-truck distance

differenceDistTruckOrder$_j$ % difference between distances
= {$\forall$ i$\in\triangle$truckOrder$_j$ $\underline{\text{dOrderTruck}_{i,j}}$ }[2] – % 2nd least order-truck distance
{$\forall$ i$\in\triangle$truckOrder$_j$ $\underline{\text{dOrderTruck}_{i,j}}$ }[1], % the least order-truck distance
),

$\forall$ i $\in$ SOURCEORDER

differenceDistSuccOrder$_i$ % difference between distances
= {$\forall$ j $\in$ $\triangle$nextOrder$_i$ dOrderOrder$_{i,j}$ } [2] – % 2nd least order-succ distance
{$\forall$ j $\in$ $\triangle$nextOrder$_i$ dOrderOrder$_{i,j}$ } [1] , % the least order-succ distance

$\forall$ j $\in$ PICKUPORDER $\rightarrow$ (
min #$\triangle$truckOrder$_j$, % least slack fewest trucks
max differenceDistTruckOrder$_j$, % least regret: easiest choice
min #$\triangle$nextOrder$_j$, % least slack
)
$\forall$ i $\in$ $\triangle$truckOrder$_j$ $\rightarrow$ (
min $\underline{\text{dOrderTruck}_{i,j}}$, % greedy search
min #$\triangle$OrderTruck$_i$ % least slack
)
j $\in$ OrderTruck$_j$?, % query on “j $\in$ OrderTruck$_i$”

$\forall$ j $\in$ SOURCEORDER $\rightarrow$ (
min #$\triangle$nextOrder$_j$, % least slack: fewest successors
max differenceDistSuccOrder$_j$, % least regret: easiest choice
min #$\triangle$truckOrder$_j$, % least slack
)
$\forall$ k $\in$ $\triangle$nextOrder$_j$ $\rightarrow$ (
min tOrderOrder$_{j,k}$ % greedy search
)
nextOrder$_j$ = k ?, % query on “nextOrder$_j$ = k”

$\forall$ i $\in$ SOURCEORDER
$t1Order_i$ = ?,

min $\sum_{i \in SOURCEORDER}$ #$TimeOrder_i$ - $\sum_{i \in SOURCEORDER}$ $serviceTimeOrder_i$.

As commented in the NCL program, general parameters include *nbTruck*, *capacityTruck* and *nbOrder*. Each order has its identifier, coordinates, demand, time window, service time, pick-up and delivery locations. If $demandOrder_i$ is non-negative, it means a pick-up order to which is associated a delivery order $idxDelivrOrder_i$; otherwise, it means a delivery order to which is associated a pick-up order $idxPickupOrder_i$. $dOrder_{i,j}$ is the Euclidean distance from order *i* to *j*; $tOrder_{i,j}$ (*service time* + $dOrder_{i,j}$) represents time distance from order *i* to *j*.

To each truck *i* is associated an origin $sourceTruck_i$, a destination $sinkTruck_i$, and order set $OrderTruck_i$, which form a tour. Truck *i* is active iff $OrderTruck_i \neq \varnothing$. For each order *i*, the model introduces successor variable $nextOrder_i$ and time interval variable $TimeOrder_i$ (work and driving time from order *i* to its successor), with $t1Order_i$ and $t2Order_i$ being starting and ending times.

Different from [1] which uses integer sorting of 3 tuples of variables to model vehicle routing and generate feasible schedules, the present model adopts set sorting of 2 tuples of variables and the routing algorithm here is exact. The modeling of main constraints is explained below:

- "Set sorting" is used to express that time interval of order *i* precedes that of its successor, and different orders have different successors:

  $\forall$ i $\in$ SOURCEORDER
  $TimeOrder_i \prec TimeOrder_{nextOrder_i}$,

  $\forall$ i $\neq$ j $\in$ SOURCEORDER
  $nextOrder_i \neq nextOrder_j$,

- For order *i*, the length of the time interval $TimeOrder_i$ equals to the time distance (service + driving) from order *i* to its successor, and the balance of the load of its truck should hold:

  $\forall$ i $\in$ SOURCEORDER (
  #$TimeOrder_i$ = $tOrder_{i,\, nextOrder_i}$,
  $loadOrder_{nextOrder_i}$ = $loadOrder_i$ + $demandOrder_{nextOrder_i}$
  ),

- Pick-up precedes delivery, and should be coupled in a same truck:

  $\forall$ i $\in$ PICKUPORDER (
  $TimeOrder_i \prec TimeOrder_{idxDelivrOrder_i}$,
  $truckOrder_i$ = $truckOrder_{idxDelivrOrder_i}$
  ),

- To break symmetry, the model requires that a truck with lower number is used in preference and contains lower supremum of the origin and orders:

$\forall$ i < j $\in$ TRUCK (

$\text{activeTruck}_j \rightarrow \text{activeTruck}_i$,

sup $\text{OrderSourceTruck}_i$ > *sup* $\text{OrderSourceTruck}_j$,

),

Logically it is sufficient to instantiate the successor variable $nextOrder_i$ for each order *i*. This is efficient when there are only a few trucks. For general purpose, search can be programmed in 2 steps: First at set partitioning level it instantiates the truck order variable $OrderTruck_i$ for each truck *i*; next at routing level it instantiates the successor variable $nextOrder_i$ for each order *i*. The instantiation of the release time variable $t1Order_i$ for all order *i* is simply a final verification for a solution.

In addition to least slack rules, the NCL model also uses regret-based search rules for solving the problem. To set up the regret criterion for set partitioning, $dOrderTruck_{i,j}$ is introduced to represent "fuzzy" distance from order *j* to truck *i*. For pick-up order *i*, $differenceDistTruckOrder_i$ is introduced to calculate the difference between the second least and the least order-truck distances. If the difference is the biggest, hopefully regret on order-truck choice is the least. To query on $OrderTruck_i$, the search rules are:

- First, select a critical order *j* in the lexicographic order of (fewest truck candidates, least regret on order-truck choice, fewest successor candidates);
- Next, for the fixed order *j*, select a truck *i* in the lexicographic order of (least truck-order distance, fewest order candidates);
- The query "$j \in OrderTruck_i$?" triggers the branching: first try to put order *j* to truck *i*; next try the inverse logic.

For any order *i*, variable $differenceDistSuccOrder_i$ is introduced to calculate the difference between the second least and the least order-successor distances. Search on $nextOrder_i$ follows the rules as below:

- First, select a critical order *j* in the lexicographic order of (fewest successor candidates, least regret on successor choice, fewest truck candidates);
- Next, for order *j*, greedily select a successor *i* with the smallest distance;
- The query "$nextOrder_j = k$?" triggers the branching: first try to constrain $order_j$'s successor to be *k*; next try the inverse logic.

The objective is to minimize the driving time: $\Sigma_{i \in SOURCEORDER} \#TimeOrder_i - \Sigma_{i \in SOURCEORDER} serviceTimeOrder_i$.

NCL can solve at least 4 problems (*LC101, LC201, LR101, LRC101*) of [2] with optimality proofs. Optimal bounds proved are respectively 829 for *LC101*, 590 for *LC201*, 1638 for *LR101* and 1702 for *LRC101*. The computation time bounds are 20

seconds for *LC101, LC201, LR101* and 3 hours for *LRC101*. The optimal solutions proved by NCL are:

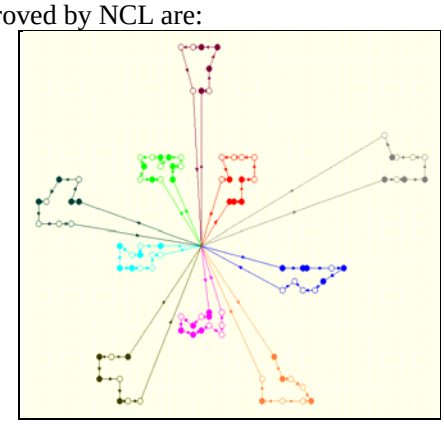

Figure 2: *LC101* (10 trucks, 106 orders), cost: 829

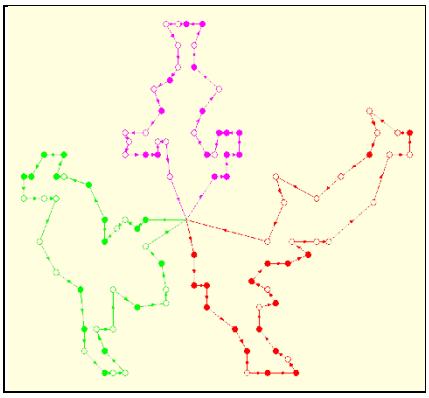

Figure 3. *LC201* (3 trucks, 102 orders), cost: 590

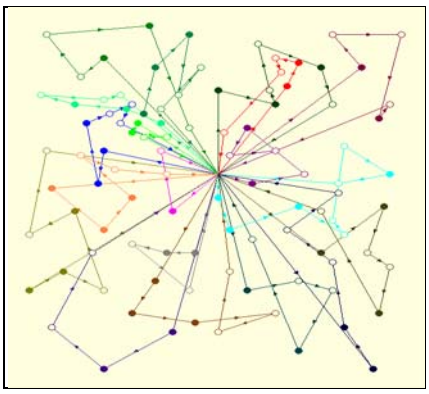

Figure 4. *LR101* (19 trucks, 106 orders), cost: 1638

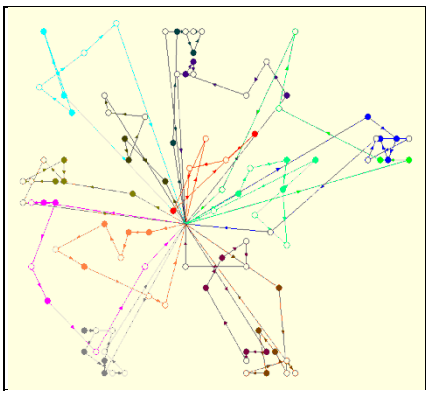

Figure 5. *LRC101* (14 trucks, 106 orders), cost: 1702

4 Conclusion

This paper presents the modeling and solving of the pick-up and delivery problem with time windows using Mixed Set Programming. A complete constraint program is given. In industrial applications, routing problems may involve many complex side constraints. Fortunately, the flexibility of MSP allows describing and solving non-linear constraints in a natural way.

References

[1] E. Domenjoud, C. Kirchner, J. Zhou. *Generating Feasible Schedules for a Pick-Up and Delivery Problem.* Technical Report of Loria, France, 1999.

[2] H. Li and A. Lim, *A MetaHeuristic for the Pick-up and Delivery Problem with Time Windows*, In Proceedings of the 13th International Conference on Tools with Artificial Intelligence, Dallas, TX, USA, 2001.
[3] T. Müller. *Solving set partitioning problems with constraint programming*. In PAPPACT98, pages 313-332. The Practical Application Company Ltd, 1998.
[4] L.-M. Rousseau, M. Gendreau, G. Pesant. *Solving VRPTWs with Constraint Programming Based Column Generation*. Annals OR 130(1-4): 199-216 (2004).
[5] M.M. Solomon. *The vehicle routing and scheduling problems with time window constraints*. Operations research, 35: 254-265, 1987.
[6] J. Zhou. *A permutation-based approach for solving the Job-shop problem*. Constraints 2(2): 185-213, (1997).
[7] J. Zhou. *Introduction to the constraint language NCL*. JLP45(1-3): 71-103 (2000).
[8] J. Zhou. *A Note On Mixed Set Programming*. Proc. of The 7th International Symposium on Operations Research and Its Applications, pp. 131-140. Lijiang, China, October 2008.
[9] J. Zhou. *The NCL Natural Constraint Language (in Chinese).* Science Press. 236 pp, 2009.