Finding the Optimal Delivery Plan: Model as a Constraint Satisfaction Problem

Radu Jurca, Nguyen Quang Huy and Michael Schumacher
Intelligent Agents Course 2006–2007

1 Problem Definition

One company owns several vehicles that are supposed to optimally deliver a set of tasks in a given network. The goal of the company is to determine a plan for each of its vehicles such that:

1. all tasks assigned to the company are delivered;
2. vehicles carry out tasks sequentially (i.e., every vehicle must first deliver the loaded task before picking up another task)
3. the total revenue of the company is maximized;

We assume that the total revenue of the company is defined as the sum of the individual revenues obtained by each vehicle. The revenue of a vehicle is computed as the sum of the rewards (obtained from delivering the tasks) minus the total cost incurred for delivering the tasks. The total cost for delivering the tasks is computed by multiplying the number of driven kilometers by the cost per kilometer of the vehicle.

Formally, let:

- $V = \{v_1, v_2, \ldots v_{N_V}\}$ be the set of vehicles owned by the company, and available for delivering the tasks;
- $T = \{t_1, t_2, \ldots t_{N_T}\}$ be the set of tasks to be delivered.

Every task is characterized by a pickup city and a delivery city. We assume we already know the shortest path between any two cities.

2 Solution

As vehicles cannot carry out tasks in parallel, a delivery plan for a vehicle is uniquely characterized by the delivery sequence of the assigned tasks. For example, if t_1 and t_2 are assigned to the first vehicle, it is enough to tell the vehicle to first carry out t_1 and then carry out t_2. The plan of the vehicle is completely determined and consists of the following actions:

- move on the shortest path to the pickup point of t_1, then
- move on the shortest path to the delivery point of t_1, then
- move on the shortest path to the pickup point of t_2, then
- move on the shortest path to the delivery point of t_2.

The optimal solution for the company consists of an ordered delivery sequence for each of its vehicle, such that the constraints are satisfied.

3 Encoding as a CSP

We use the following variables to define a constraint satisfaction (optimization) problem that finds the optimal plan for a company:

- $nextTask$ - an array of N_T+N_V variables. The array contains one variable for every existing task, and one variable for every existing vehicle:

  $$nextTask = [nextTask(t_1), \ldots, nextTask(t_{N_T}), nextTask(v_1), \ldots, nextTask(v_{N_V})];$$

  One variable from the $nextTask$ array can take as a value another task, or the value $NULL$:

  $$nextTask(x) \in \{t_1, t_2, \ldots, t_{N_T}, NULL\};$$

  with the following semantics:

  - if $nextTask(t_i) = t_j$ it means that some vehicle will deliver the task t_j immediately after delivering t_i;
  - if $nextTask(v_k) = t_j$ it means that the vehicle v_k first delivers the task t_j;
  - if $nextTask(t_i) = NULL$, the vehicle that delivered the task t_i does not have to deliver any other tasks;
  - if $nextTask(v_k) = NULL$, the vehicle v_k does not have to deliver any task.

- $time$ - an array of N_T variables. The array contains one variable for every existing task.

  $$time = [time(t_1), \ldots, time(t_{N_T})];$$

  One variable from the $time$ array can take an integer value specifying the delivery sequence number of the task in the plan of a certain vehicle:

  $$time(x) \in \{1, 2, \ldots, N_T\};$$

  We therefore have:

- if $nextTask(v_k) = t_j$, the task t_j is the first to be delivered by the vehicle v_k and therefore $time(t_j) = 1$;
- if $nextTask(t_i) = t_j$, the task t_j is delivered immediately after the task t_i by some vehicle, and therefore, $time(t_j) = time(t_i) + 1$;

- *vehicle* - a redundant array of N_T variables, one variable for each task:

$$vehicle = [vehicle(t_1), \ldots, vehicle(t_{N_T})];$$

One variable from the *vehicle* array can take as a value the code of the vehicle that delivers the corresponding task:

$$vehicle(x) \in \{v_1, v_2, \ldots, v_{N_V}\};$$

such that:

- if $nextTask(v_k) = t_j$, the task t_j is the first to be delivered by the vehicle v_k and therefore $vehicle(t_j) = v_k$;
- if $nextTask(t_i) = t_j$, the task t_j is delivered immediately after the task t_i by some vehicle, and therefore, $vehicle(t_j) = vehicle(t_i)$;

A valid plan for the company (i.e., a set of plans for each of its vehicles) is a value allocation for each of the above variables that satisfies the following constraints:

3.1 Constraints

1. $nextTask(t) \neq t$: the task delivered after some task t cannot be the same task;
2. $nextTask(v_k) = t_j \Rightarrow time(t_j) = 1$: already explained;
3. $nextTask(t_i) = t_j \Rightarrow time(t_j) = time(t_i) + 1$: already explained;
4. $nextTask(v_k) = t_j \Rightarrow vehicle(t_j) = v_k$: already explained;
5. $nextTask(t_i) = t_j \Rightarrow vehicle(t_j) = vehicle(t_i)$: already explained;
6. all tasks must be delivered: the set of values of the variables in the $nextTask$ array must be equal to the set of tasks T plus N_V times the value $NULL$.
7. the capacity of a vehicle cannot be exceeded: if $load(t_i) > capacity(v_k) \Rightarrow vehicle(t_i) \neq v_k$

3.2 Objective

From all of the valid plans (i.e. value allocations that satisfy the above constraints) we search for the optimal one: i.e., the plan that maximizes the revenue of the company. All tasks need to be delivered; as the reward for one task is constant (and defined in the XML file), the total reward of the company is constant. We therefore want to find the plan that minimizes the cost.

The cost of the company is defined as the sum of the costs incurred by individual vehicles for carrying out the assigned tasks. The cost of one vehicle is defined by the distance travelled by the vehicle, multiplied by the cost per kilometer.

Before defining the cost, let us introduce the following notation:

- $dist(t_i, t_j)$ is the shortest distance between the delivery point of the task t_i and the pickup point of the task t_j;
- $dist(t_i, NULL) = 0$; the vehicle stops after delivering the task t_i;
- $dist(v_k, t_j)$ is the shortest distance between the home location of the vehicle v_k and the pickup point of the task t_j;
- $dist(v_k, NULL) = 0$;
- $length(t_i)$ is the shortest distance from the pickup point to the delivery point of the task t_i;
- $length(NULL) = 0$;
- $cost(v_k)$ is the cost per kilometer of the vehicle v_k;

The total cost of the company is defined as:

$$C = \sum_{i=1}^{N_T} \Big(dist\big(t_i, nextTask(t_i)\big) + length\big(nextTask(t_i)\big) \Big) \cdot cost\big(vehicle(t_i)\big) + \sum_{k=1}^{N_V} \Big(dist\big(v_k, nextTask(v_k)\big) + length\big(nextTask(v_k)\big) \Big) \cdot cost\big(v_k\big); \quad (1)$$

The optimal plan for the company results from the value allocation to the defined variables such that the constraints are satisfied and the cost C is minimized.

4 Stochastic Local Search algorithm for COP

A discrete constraint optimization problem (COP) is a tuple $< X, D, C, f >$ where:

- $X = \{x_1, .., x_n\}$ is a set of variables.

- $D = \{d_1, .., d_n\}$ is a set of domains of the variables, each given as a finite set of possible values.
- $C = \{c_1, .., c_p\}$ is a set of constraints, where a constraint c_i is a function $d_{i1} \times .. \times d_{il} \rightarrow \{0, 1\}$ that returns 1 if the value combination is allowed and 0 if it is not.
- $f : d_1 \times ... \times d_n \rightarrow \Re$ is the objective function that we want to minimize (or maximize).

The optimal solution of a COP is an assignment of values to all variables that satisfies all constraints and minimizes the objective function.

The stochastic local search (SLS) is an algorithm that searches for a sub optimal but close to the global optimal solution for a COP. More details of the algorithm can be found in [1]. The idea of SLS is simple: the local search process is started by selecting an initial candidate solution, and then proceeds by iteratively moving from one candidate solution to a neighbouring candidate solution, where the decision in each search step is made stochastically and based on a limited amount of local information only. A sketch of SLS algorithm is given in the algorithm 1.

Algorithm 1 *SLS algorithm for COP*

procedure *SLS(X,D,C,f)*
  $A \leftarrow SelectInitialSolution(X, D, C, f)$
  repeat
    $A^{old} \leftarrow A$
    $N \leftarrow ChooseNeighbours(A^{old}, X, D, C, f)$
    $A \leftarrow LocalChoice(N, f)$
  until termination condition met
  return A
end procedure

SelectInitialSolution(): This function selects a complete, possibly random, assignment A of values to all variables that is consistent with the constraints.

ChooseNeighbours(): This function provides a set of candidate assignments that are close to the current one and could possibly improve it. In the simple case, they are generated by randomly selecting a variable $x_i \in X$ and generating all assignments that are equal to A but assign to x_i different values in the domain of x_i that are consistent with the rest of A according to the constraints.

LocalChoice(): It first selects the assignment A in the set of candidates that gives the best improvement of the objective function. If there are multiple equally good assignments, it chooses one randomly. Then with probability p it returns A, with probability $1 - p$ it returns the current assignment A^{old}. The probability p is a parameter of the algorithm. If p is close to 1, the algorithm converges faster but it is easily trapped into a local minima. A value of p from 0.3 to 0.5 would be a good choice.

The iteration continues until a termination condition is met, for example when there is no further improvement for some number of steps or the maximum number of steps is reached.

5 Applying SLS algorithm on Optimal Delivery Plan problem

The discussion below assumes that vehicle can only carry one task at a time. You need to take into account the possibility of vehicle carrying multiple tasks. Therefore, the description below can only serve as a guidelines how to implement SLS for the extended problem.

In this section we implement the SLS algorithm given in the algorithm 1 for the CSP formulation in section 3. In our CSP encoding we have

- $X = \{nextTask, time, vehicle\}$ is the set of variables ($n = 3N_T + N_V$ variables).
- $D = \{d_1, .., d_n\}$ is the set of domains where
  $d_{nextTask(i)} = \{t_1, t_2, \ldots, t_{N_T}, NULL\}$ for $i = t_1..t_{N_T}$ or $i = v_1..v_{N_V}$;
  $d_{time(t_i)} = \{1, 2, \ldots, N_T\}$ for $i = t_1..t_{N_T}$;
  $d_{vehicle(t_i)} = \{1, 2, \ldots, N_T\}$ for $i = t_1..t_{N_T}$;
- $C = \{c_1, .., c_p\}$ is the set of constraints given in the section 3.1.
- f is the objective function given by the equation 1.

The corresponding functions in the algorithm 1 are defined in the following sections. Some of the functions have already the full pseudo-code. For others, you must define them yourself.

5.1 SelectInitialSolution() function:

For a simple initial solution, we give all the tasks to the biggest vehicle. If there exist some tasks that do not fit for the vehicle, then the problem is unsolvable.
function *SelectInitialSolution(X,D,C,f)*
  // TO DEFINE
end function

5.2 ChooseNeighbours() function:

As the variables in our CSP encoding are correlated, we will use the two following local operators for finding the neighbours of the current solution:

- *Changing vehicle*: take the first task from the tasks of one vehicle and give it to another vehicle.

- *Changing task order*: change the order of two tasks in the task list of a vehicle.

In each iteration, we choose one vehicle at random and perform local operators on this vehicle to compute the neighbour solutions. The pseudo-code for the function is the following:

function *ChooseNeighbours*(A^{old},X,D,C,f)
  $N = \{\}$
  $v_i = random(v_1..v_{N_V})$ such that $A^{old}_{nextTask(v_i)} \neq NULL$
  // Applying the *changing vehicle* operator :
  for $v_j \in (v_1..v_{N_V}), v_j \neq v_i$ **do**
    $t = A^{old}_{nextTask(v_i)}$
    if $load(t) \leq capacity(v_j)$ **then**
      $A = ChangingVehicle(A^{old}, v_i, v_j)$
      $N = N \cup \{A\}$
    end if
  end for
  // Applying the *Changing task order* operator :
  // compute the number of tasks of the vehicle
  $length = 0$
  $t = v_i$ // current task in the list
  repeat
    $t = A^{old}_{nextTask(t)}$
    $length = length + 1$
  until $t = NULL$
  if $length \geq 2$ **then**
    for $tIdx_1 \in (1..length - 1)$ **do**
      for $tIdx_2 \in (tIdx_1 + 1..length)$ **do**
        $A = ChangingTaskOrder(A^{old}, v_i, tIdx_1, tIdx_2)$
        $N = N \cup \{A\}$
      end for
    end for
  end if
  return N
end function

Where $ChangingVehicle()$ and $ChangingTaskOrder()$ are the functions corresponding to the local operators.

function *ChangingVehicle*(A, v_1, v_2)
  $A1 = A$
  $t = nextTask(v_1)$
  $A1_{nextTask(v_1)} = A1_{nextTask(t)}$
  $A1_{nextTask(t)} = A1_{nextTask(v_2)}$
  $A1_{nextTask(v_2)} = t$
  $UpdateTime(A1, v_1)$

$UpdateTime(A1, v_2)$
$A1_{vehicle(t)} = v_2$
return $A1$
end function

function *ChangingTaskOrder*$(A, v_i, tIdx_1, tIdx_2)$
$A1 = A$
$tPre_1 = v_i$ // previous task of $task_1$
$t_1 = A1_{nextTask(tPre_1)}$ // $task_1$
$count = 1$
while $count < tIdx_1$ **do**
$tPre_1 = t_1$
$t_1 = A1_{nextTask(t_1)}$
$count++$
end while
$tPost_1 = A1_{nextTask(t_1)}$ // the task delivered after t_1
$tPre_2 = t_1$ // previous task of $task_2$
$t_2 = A1_{nextTask(tPre_2)}$ // $task_2$
$count++$
while $count < tIdx_2$ **do**
$tPre_2 = t_2$
$t_2 = A1_{nextTask(t_2)}$
$count++$
end while
$tPost_2 = A1_{nextTask(t_2)}$ // the task delivered after t_2
// exchanging two tasks
if $tPost_1 = t_2$ **then**
// the task t_2 is delivered immediately after t_1
$A1_{nextTask(tPre_1)} = t_2$
$A1_{nextTask(t_2)} = t_1$
$A1_{nextTask(t_1)} = tPost_2$
else
$A1_{nextTask(tPre_1)} = t_2$
$A1_{nextTask(tPre_2)} = t_1$
$A1_{nextTask(t_2)} = tPost_1$
$A1_{nextTask(t_1)} = tPost_2$
end if
$UpdateTime(A1, v_i)$
return $A1$
end function

UpdateTime() is a function that updates the values of the variables in vector *time* that associates with a vehicle v_i of a complete assignment:

procedure *UpdateTime*(A, v_i)
$t_i = A_{nextTask(v_i)}$

if $t_i \neq NULL$ **then**
$A_{time(t_i)} = 1$
repeat
$t_j = A_{nextTask(t_i)}$
if $t_j \neq NULL$ **then**
$A_{time(t_j)} = A_{time(t_i)} + 1$
$t_i = t_j$
end if
until $t_j = NULL$
end if
end procedure

5.3 LocalChoice() function:

procedure *LocalChoice(N,f)*
TO DEFINE
end procedure

5.4 The termination condition:

The search process terminates when a maximum number of iterations is reached. We can set this number to 10000 iterations or more depends on the solution quality and the problem size.

References

[1] Zhang, Weixiong and Wang, Guandong and Xing, Zhao and Wittenburg,Lars: "Distributed stochastic search and distributed breakout: properties, comparison and applications to constraint optimization problems in sensor networks," *Artificial Intelligence* **161**(1-2), pp. 55-88, 2005.