Capacitated Vehicle Routing Problem (cvrp2)

—- Run LPL Code , HTML Document –

Problem: A fixed fleet of delivery vehicles of uniform capacity must service known customer demands for a single commodity from a common depot at minimum transit cost. An concrete application is given in cvrp[1]. The models cvrp-1[2], cvrp-2[3], and cvrp-3[4] give MIP-formulations of the problem. Formulate this problem as a permutation problem[5].

Modeling Steps

Let $i, j \in I = \{1, \ldots, n-1\}$ be a set of customer locations (n is the number of locations including the warehouse (or the depot) and let $k \in K = \{1, \ldots, m\}$ be a set of trucks. Let the capacity of each truck be CA, and let the demand quantity to deliver to a customer i be dem_i. Furthermore, the distance between two customer i and j is $d_{i,j}$. Finally, the distance from the warehouse – the depot from where the trucks start – to each customer i is dw_i.

The customers are enumerated with integers from 1 to $n-1$. If there were one single truck (that must visits all customers), every permutation sequence of the numbers 1 to $n-1$ would define a legal tour. Since we have m trucks, a sequence of a subset of 1 to $n-1$ has to be assigned to each truck. Hence, each truck starts at the depot, visiting a (disjoint) subset of customers in a given order and returns to the depot.

The variables can now be formulated as a "partitioned permutation". Example: if we had 3 trucks and 10 customers then the 3 subset sequences:

$$\{\{2,3,4\},\{1,9,8,5\},\{6,7,10\}\}$$

defines a partitioning between the 3 trucks. It means, for example, that truck 1 starts at the depot visiting customers 2, 3, 4 in this order and returns to the depot. In LPL, we can declare these partitioned permutation with a permutation variable $x_{k,i} \in [1 \ldots n-1]$. For example: $x_{1,1} = 2$ means that the customer number 2 is visited by the truck 1 right after the depot, $x_{1,2} = 3$ means that the next customer (after 2) is 3, etc.

The unique constraint is the load of the truck that these subsets must fullfill (each subset sequence must be chosen in such a way that the capacity of the truck is larger than the cumulated demand of the customers that it visits):

$$\sum_{i|x_{k,i}} dem_{x_{k,i}} \leq CA \quad \text{forall } k \in K$$

We want to minimize the total travel distance of the trucks. Let cc_k be the size of the subset k (the numbers of customers that the truck k visits). Of course that number is variable and cannot be given in advance! Let $routeDistances_k$ be the (unknown) travel distance of truck k, then we want to minimize the total distances (and in a first round the number of trucks):

$$\min \sum_k routeDistances_k$$

[1] https://lpl.matmod.ch/lpl/Solver.jsp?name=/cvrp
[2] https://lpl.matmod.ch/lpl/Solver.jsp?name=/cvrp-1
[3] https://lpl.matmod.ch/lpl/Solver.jsp?name=/cvrp-2
[4] https://lpl.matmod.ch/lpl/Solver.jsp?name=/cvrp-3
[5] For a definition of *permutation problems* see [2].

where[6]

$$routeDistances_k = \sum_{i \in 2..cc_k} d_{x_{k,i-1},x_{k,i}} + \text{if}(cc_k > 0, dw_{x_{k,1}} + dw_{x_{k,cc_k}}) \quad \text{forall } k \in K$$

The term $dw_{x_{k,1}}$ denotes the distance of the truck k from the depot to the first customer, and $dw_{x_{k,cc_k}}$ is the distance of the truck tour k from the last customer to the depot, $d_{x_{k,i-1},x_{k,i}}$ is the distance from a customer to the next – namely from customer $x_{k,i-1}$ to customer $x_{k,i}$ – and $\sum_{i \in 2..cc_k} \ldots$ sums that distances of a tour k.

Further Comments: One of the most distiguished feature of LPL and the LSP (LocalSolver language) is the use of variables within passive indices. This allows one to formulate an entire and practical important class of problems (the permutation problems, see my Permutation Paper). No other modeling system, to my knowledge, has this feature.

A model in LSP is partitioned into, basically 4 functions: *input()*, where the data input is defined, *model()*, to declare the modeling structure, *param()* to specify the solver parameters, and *output()*, a procedure that describes the output.

Apart of *real*, *integer*, and *boolean* variables, LSP contains the *list* and *set* variables to specify permutations (see my paper above). In LPL, this kind of variables is declared as 1- and 2-dimensional integer variables with the keyword `alldiff`. Note also the graphical output of LPL and the concise code for reading the data.

[6]The expression $if(boolExpr, Expr)$ returns $Expr$ if $boolExpr$ is true else it returns 0 (zero).

LPL code (run cvrp2)

```
model cvrp "Capacitated Vehicule Routing Problem";
  set i,j  "customers";
      k    "trucks";
  parameter
      d{i,j} "distances";
      dw{i}  "distance from/to warehouse";
      dem{i} "demand";
      CA     "truck capacity";
  alldiff x{k,i} 'partition' "customerSequences";
  expression
    cc{k}: count{i} x;
    trucksUsed{k}: cc[k] > 0;
    routeDistances{k}:
       sum{i in 2..cc} d[x[k,i-1],x[k,i]]
       + if(cc[k]>0, dw[x[k,1]] + dw[x[k,cc[k]]]);
    nbTrucksUsed: sum{k} trucksUsed[k];
    totalDistance: sum{k} routeDistances[k];
  constraint CAP{k}: sum{i|x} dem[x[k,i]] <= CA;
  minimize obj1: nbTrucksUsed;
  minimize obj2: totalDistance;
end
```

LocalSolver code (download cvrp.lsp)

```
use io;
function input() {
  readInputCvrp();
  if (nbTrucks == nil) nbTrucks = getNbTrucks();
  computeDistanceMatrix();
}

function model() {
  customersSequences[k in 1..nbTrucks] <- list(
      nbCustomers);
  constraint partition[k in 1..nbTrucks](
      customersSequences[k]);
  for [k in 1..nbTrucks] {
    local sequence <- customersSequences[k];
    local c <- count(sequence);
    trucksUsed[k] <- c > 0;
    routeQuantity <- sum(0..c-1, i => demands[
        sequence[i]]);
    constraint routeQuantity <= truckCapacity;
    routeDistances[k] <- sum(1..c-1, i =>
        distanceMatrix[sequence[i - 1]][sequence[i]])
      + (c > 0 ? (distanceWarehouse[sequence[0]] +
          distanceWarehouse[sequence[c - 1]]) : 0);
  }
  nbTrucksUsed <- sum[k in 1..nbTrucks](trucksUsed[k
      ]);
  totalDistance <- sum[k in 1..nbTrucks](
      routeDistances[k]);
  minimize nbTrucksUsed;
  minimize totalDistance;
}
```

LPL input data model for cvrp2

```
model data;
  set h; //h is i plus 1, 1 is depot (warehouse)
  parameter de{h}; n;m; X{h};Y{h}; string typ; dum;
  Read('A-n32-k5.vrp,%1;-1:DIMENSION',dum,dum,n);
  Read('%1;-1:EDGE_WEIGHT_TYPE',dum,dum,typ);
  if typ<>'EUC_2D' then
    Write('Only␣EUC_2D␣is␣supported\n'); return 0;
  end;
  Read('%1;-1:CAPACITY',dum,dum,CA);
  Read{h}('%1:NODE_COORD_SECTION:DEMAND_SECTION',
      dum,X,Y);
  Read{h}('%1:DEMAND_SECTION:DEPOT_SECTION',dum,de);
  m:=Ceil(sum{h} de/CA);
  k:=1..m;
  i:=1..n-1;
  d{i,j}:=Round(Sqrt((X[i+1]-X[j+1])^2+(Y[i+1]-Y[j
      +1])^2));
  dem{i}:=de[i+1];
  dw{i}:=Round(Sqrt((X[i+1]-X[1])^2+(Y[i+1]-Y[1])^2)
      );
end
```

LPL output model for cvrp2

```
model output friend data;
  parameter y{k,i};
  {k} (y[k,1]:=1, y[k,cc+2]:=1,{i|i<=cc} (y[k,i+1]:=
      x+1));
  Draw.Scale(5,5);
  {k,i in 1..cc+1} Draw.Line(X[y],Y[y],X[y[k,i+1]],Y
      [y[k,i+1]],k+3,3);
  {k,i in 1..cc+2} Draw.Circle(y&'',X[y],Y[y],2,1,0)
      ;
end
```

LocalSolver model input code

```
function param() { .... }
function output() { ....  }

function readInputCvrp() {
  local inFile = io.openRead(inFileName);
  local nbNodes = 0;
  while (true) {
    local str = inFile.readString();
    if(str.startsWith("DIMENSION")) {
      if (!str.endsWith(":")) str=inFile.readString
          ();
      nbNodes = inFile.readInt();
      nbCustomers = nbNodes - 1;
    } else if ((str.startsWith("CAPACITY"))) {
      if(!str.endsWith(":")) str=inFile.readString()
          ;
      truckCapacity = inFile.readInt();
    } else if((str.startsWith("EDGE_WEIGHT_TYPE")))
        {
      if(!str.endsWith(":")) str=inFile.readString()
          ;
        local weightType = inFile.readString();
        if (weightType != "EUC_2D") throw ("Edge␣
            Weight␣Type␣" + weightType + "␣is␣not␣
            supported␣(only␣EUC_2D)");
    } else if(str.startsWith("NODE_COORD_SECTION"))
        {
        break;
    } else { local dump = inFile.readln(); }
  }
  //nodeX and nodeY are indexed by original data
      indices (1 for depot)
  for[n in 1..nbNodes] {
    if (n != inFile.readInt()) throw "Unexpected␣
        index";
```

The output graph of LPL

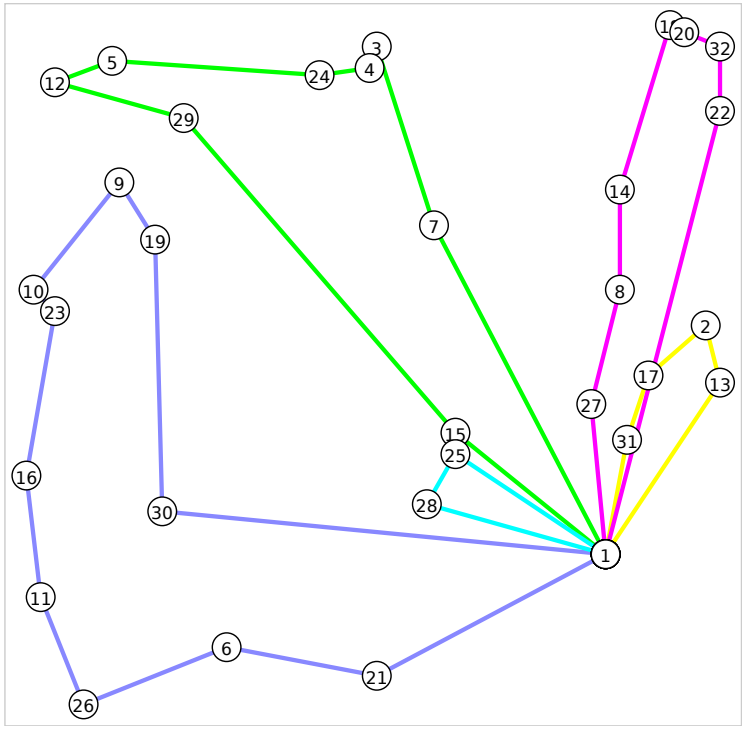

Figure 1: Optimal Solution of “A-n32-k5.vrp”

LocalSolver input code (continued)

```
    nodesX[n] = round(inFile.readDouble());
    nodesY[n] = round(inFile.readDouble());
  }
  dump = inFile.readln();
  if (!dump.startsWith("DEMAND_SECTION")) throw "
      Expected␣keyword␣DEMAND_SECTION";
  for[n in 1..nbNodes] {
    if(n!=inFile.readInt()) throw "Unexpected␣index"
        ;
    local demand = inFile.readInt();
    if (n == 1) {
      if (demand != 0) throw "expected␣demand␣for␣
          depot␣is␣0";
    } else {
      demands[n-2] = demand; // demands is indexed
          by customers
    }
  }
  dump = inFile.readln();
  if (!dump.startsWith("DEPOT_SECTION")) throw "
      Expected␣keyword␣DEPOT_SECTION";
  local warehouseId = inFile.readInt();
  if (warehouseId != 1) throw "Warehouse␣id␣is␣
      supposed␣to␣be␣1";
  local endOfDepotSection = inFile.readInt();
  if (endOfDepotSection != -1) throw "Expecting␣only
      ␣one␣warehouse,␣more␣than␣one␣found";
}

function computeDistanceMatrix() { .... }
function getNbTrucks() { .... }
```

References

[1] MatMod. Homepage for Learning Mathematical Modeling : https://matmod.ch.

[2] Hürlimann T. Various Model Types. https://matmod.ch/lpl/doc/variants.pdf.