Algorithmic Methods for Mathematical Models:

Course Project

Chetan K.C

2016-06-22

Contents

Problem statement

In this problem we are dealing with the no of packets and the number of trucks with considering the size of the packets. The packets are the rigid bodies so they can't be rotated so that we can't put 3x4 packet in 2x5 they are design by this way, And another important thing that we need to consider is that one packet couldn't put above the another one. Due to some specific design two packets couldn't put in the same packets if [p1][p2] = 1. we have to design ilp module considering these very important things and the more things to be consider is we can't left the packets (i.e: we need to load each packet in one of the truck in only one of the truck).

The objective function is to distribute the packages among the trucks such a way that we will use less trucks with the minimum load as possible, so we could reduce the number of trucks used.

Integer Linear model

Notations

Following notations are used to model the given problem.

Indices

X=1..xtruck;

Y=1..ytruck;

T=1..nt;

P=1..np;

Sets

P set of packets

T set of trucks

Parameters

xtruck = x-dimension of the truck.

ytruck = y-dimension of the truck.

wtruck = capacity of the truck.1

nt=number of trucks.

np=number of packets.

xDim=x-dimension of the packet.

yDim=y-dimension of the packet.

wp= weight of the packet.

SDM= incompatible matrix.

Decision variables

dvar boolean Pt[p in P][t in T]; *//packet is either in the truck t or not*

dvar boolean Pxy[p in P][x in X][y in Y]; *//package p is in the cell with upper-right coordinates.*

dvar boolean Pbl[p in P][x in X][y in Y]; *// the bottom lñeft cell of p has upper-right Coordinates.*

dvar boolean used[t in T]; //Either the truck t is used or not

dvar int z; // to calculate the objective function.

Objective function

Objective function is to minimize the total cost, and also minimize the load in the trucks with the highest load and the no. of trucks to be used.

Minimize $z+\sum$(t in T) used[t] * wtruck

Constraints

The following constraints must be apply to the above variables to get the solution for the defined problem:

1. One packet can be assigned to only one truck.

$$\forall\ p \in P \qquad \sum (t \text{ in } T) \quad pt[p][t]==1;$$

2. Total weight of the packet in a particular truck can't be exceed the total capacity of the truck.

$$\forall\ t \in T \qquad \sum (p \text{ in } P)\ wp[p] * Pt[p][t] <= wtruck;$$

3. One packet can't put above the other.

$$\forall\ t \in T\ \forall\ x \in X\ \forall\ y \in Y\ \forall\ p1 \in P\ \forall\ p2 \in P$$

if(p1 != p2)

Pxy[p1][x][y]+Pbl[p1][x][y]+Pxy[p2][x][y]+Pbl[p2][x][y] <=3;

4. For some case packet [p1] and the packet [p2] can't be on the same truck.

$$\forall\ t \in T\ \forall\ p1 \in P\ \forall\ p2 \in P \quad Pt[p1][t]+Pt[p2][t]+SDM[p1][p2]<=2;$$

5. No packets could put in the unused truck.

$$\forall\ t \in T\ \forall\ p \in P \quad Pt[p][t]<= used[t];$$

6. Fitting of packet in the truck.

$$\forall\ p \in P$$

$\sum$(x in X) x > xtruck-xDim[p]+1) $\sum$(y in Y) y > ytruck-yDim[p]+1)

Pbl[p][x][y]==0;

$\sum$(x in X) x <= xtruck-xDim[p]+1) $\sum$(y in Y) y <= ytruck-yDim[p]+1)

Pbl[p][x][y]==1;

7. Maximum loaded truck.

$\forall$ t $\in$ T z >= $\sum$(p in P) Pt[p][t]*wp[p];

8. In every truck checking the position of pxy with respect to the pbl

$\forall$ p$\in$ P $\forall$ x $\in$ X $\forall$ y $\in$ Y

$\sum$(x1 in X) x1 >= xtruck-x && x1 <= x $\sum$(y1 in Y) y1 >= ytruck-x && y1 <= y

Pbl[p][x1][y1] == Pxy[p][x][y];

9. For all packet pbl must be equal to 1.

$\forall$ p$\in$ P $\sum$(x in X) $\sum$(y in Y) Pbl[p][x][y] ==1;

Metaheuristic solutions

GRASP

Constructive phase

```
procedure CONSTRUCTIVEPHASe (ProblemData data, double alpha)
ArrayList <candidate> candidates <- initialiseCandidates();
ArrayList <Candidate> RCL;
GraspSolution w <- newGraspSolution(this);
int qMin, qMax;
candidate c;
loop:

candidates <- evaluateArray(candidates,w)
candidates <- removeNotFeasibleCandidates(candidates);
qMin <- min(candidates);
qmax <- max(candidates);

RCL <- chooseBestElements(candidates,qmin,qMax,alpha);
c <- choose AtRandom(RCL);
w.addElement(c);
candidates.remove(c);
if(!isSolution(w)) then
goto loop:
return w;
```

The algorithm creates an empty list of candidates and then fills it with all the feasible candidates. After that, it creates an empty solution. In the loop, it evaluates each of the candidates and removes those candidates that are not feasible, this decreases the time spent iterating the candidates. Next, it fills the Restricted Candidate List (RCL) with only those candidates that are under a threshold (as the objective is to minimise). After that, one element from the RCL is selected at random, added to the solution, and removed from the candidate list. Repeat until a complete solution is made.

Local search for GRASP

```
procedure LOCALSEARCH(GRASPSOLUTION W)
boolean localOptimalSolution <- false;
loop;
        GraspSolution w1 <- findBetterSolution(w);
        if(w.getCost()==w1.getCost()then)
                localOptimalSolution <- True;
        else
                localOptimalsolution <- False;
w==w1;
if(LocalOptimalSolution)then
        goto loop;
return w;
```

In this algorithm, given a solution W, the method find Better Solution tries to find a better solution than the current one. If the new solution’s cost is the same as the cost of the previous solution, it decided that it has reached a local optimum and stops searching. If not, it keeps looking for a better solution.

The method find Better Solution is the following one:

Local search for GRASP:

```
Result: findBetterSolution(GraspSolution w)
int packets, trucks, wTruck, wPacket, xTruck, yTruck, xDim, yDim;
Random r ← newRandom();
GraspSolution w1;
loop:
        trucks ← r.nextInt(data.getnPackets());
OriginLoop:
        subtractFrom ← r.nextInt(data.getnPackets());
if wTruck >= wPacket then
 | w.truck ← packets.addPacket();
else
    ...
end if
if xTruck-xDim >= 0 and yTruck-yDim >= 0 then
 | xTruck ← xTruck − xDim;
 | yTruck ← yTruck − yDim;
 | w.truck ← packets.addPacket();
else
    ...
end if
w1=w;
w1.addValueToX_pt(packets, trucks, wPacket);
w1.addValueToX_pt(packets, subtractFrom, −wPacket);
w1.updateVariables(packets, subtractFrom, trucks);
GraspSolution w1 ← findBetterSolution(w);
return w;
```

Algorithm 3: Local search for GRASP

Greedy function for GRASP

The greediness of the algorithm lies in two places: the method where is used the formula to fill the RCL and in the main loop of GRASP.

```
procedure ChooseBestElements(Arraylist<packets> packets, int, qmin,qmax, double
Alphe)
List RCL<- NEW lIST();
Double threshold <- qmin + (qmax-qmin)* alpha;
int cNum<- 0;
LOOP;
        if (Trucks.get(i).getload()<= threshold)then
        RCL.add(c);
cNum++;
if(cNUm< packets.size()then)
goto loop;
```

```
return RCL;
```

Greedy Function for GRASP

```
Procedure SolveGRASP()
GraspSolution w, w1<- new GraspSolution();
double alpha <- 1;
loop:
        W<- CONSTRUCTIVEpHASE(DATA,ALPHA);
        W<- LocalSearch(w);
if(w.getCost()> w1.getCost()) then
w1<-w;
i<-0;
else
i<- i+1;
if(iteration == 0) then
alpha <- 0.7;
if(i==25) then
alpha<- 0.3;
        (i< MAX_NO_CHANGE)
goto loop
return w1;
```

Equation describing the RCL

The equation describing the RCL is the following one:

$RCL = \{\forall tT \mid c.getCost() \leq qMin + (qMax - qMin) \cdot \alpha)\}$

where:

• c are the candidates

• c.getCost() returns the cost of a candidate

• qMin is the value of the minimum cost among the candidates

• qMax is the value of the maximum cost among the candidates

• α is the value that represents the greediness of the algorithm. At iteration 0 it has value 1, for a pure
random solution. Next iteration, alpha gets 0.7, to allow some greediness. At iteration 25 with no
change, alpha takes value 0.3, coming close to the total greediness

BRKGA

Chromosome structure

chromosome structure:
the proposed chromosome structure is as following:

G1	G2	G3	G4	………..	Gn

the chromosome is an array of integers of size ntrucks . npackets. each of the genes takes the value of the position in the truck placed by the packet p in the truck t.

the formula that matches Xpt and the chromosome is the following one:

X_pt[p][t] = chromosome [wp.npackets+t]

Decoder

#decoder for BRKGA *procedure DECODE(CHROMOSOME CHROMOSOME)*

```
BrkgaSolution w <- new BrkgaSolution();
int NPacket <- data.getnPackets();
int NTrucks <- data.getntrucks();
usedt <- new boolean [];
X_pt <- new int [Npacket][NTruks];
b_pt <- new boolean [Npacket][Ntruks];
int p <- 0;
int t <- 0;
packetLoop
truckLoop
X_pt[p][t] <- chromosome.get(wp*Npacket+t);
b_pt[p][t] <- X_pt[p][t] != 0;
p<- p+1;
if(p < npackets) then
 goto packetLoop
else
        int p <- 0;
used_t[t] <- Helpers.sumRow(X_pt, Ntrucks,p)!=0;
t <- t+1;
if (t < ntrucks)then
goto truckloop
else
        intcost <- 0;
        cost <- ComputeCost();
w.setB_co(b_pt);
w.setCost(cost);
w.setUsed_t(used_t);
w.setX_co(X_pt);
return w;
```

Comparative results

Results from the cplex:

cplex solutions							
	capacity of truck	no.packet	no.truck	used truck	max loaded truck(Z)	sol.with OBJ	time
	7	7	5	3	6	27	00:00:00:95
	50	30	25	11	49	549	00:00:16:60
	35	20	15	12	33	383	00:00:06:91
	30	20	15	11	30	360	00:03:43:97
	90	80	25	21	90	1350	00:4:30:50
	80	80	20	16	78	1356	00:44:35:33
	80	70	20	17	77	1358	00:14:52:77
	50	40	25	19	49	549	00:00:16:15
	45	40	25	19	40	680	01:23:33:21
	45	40	30	20	45	675	00:41:55:85
	40	30	25				solution take more than two hour
	40	30	20				solution take more than two an half hour

Result from the GRASP

Result from the BRKGA

Sources and instructions

ILP is implemented in cplex, Data is self generated and will check the solution by default setting of cplex.

All the results are generated by running different datas in the cplex, if you want to test them i can provide them.

For the heuristic part i have included the psuedo code in the report but the python code is not giving me correct result so i haven’t include them.