MiniZinc Report

A solver may waste a lot of effort on gazillions of (partial) non-solutions that are symmetric to already visited ones, whereas a found solution can be transformed without search into a symmetric solution in polynomial time.

Then, when the symmetry of a problem is found, the efficiency of solving the problem can be greatly improved. In this question, there are two main symmetries.

1 Model symmetries: all resources of the same type

Assuming that a drone needs to land, selecting land Pad1 or land Pad2 has no effect on the selection of the final shipping order, that is, selecting land Pad1 or land Pad2 has no difference. Suppose there are N land pads, then land pads can be permuted: N! variable symmetries, and all these permutations preserve solutions.

In the question, there are two cases: first, the numbers of the same kind of resources used by drones using the same kind of resources and the same time period are symmetrical. Second, when resources are abundant, drones can choose any number of a resource.

Therefore, this static symmetry breaking allows us to eliminate symmetric solutions through constraints. The following constraints simplify the resource allocation of drones: for drones with the same start time, each value in the resource is less than the next one.

```
constraint forall (d1,d2 in DRONE, a in
{LAND,INSPECT,FULLSERVICE,RECHARGE,PACK} where(d1<d2)/\
     (start[d1,enum_next(ACTION,a)]=start[d2,enum_next(ACTION,a)])/\(start[d1
,a]=start[d2,a]))(
     resource[d1,a]< resource[d2,a]);
```

Another way is to stipulate that when there is no competition for resources, the drone with small number will give priority to the resources with small number.

2 Instance symmetries : drones in the same state

Drones in the same state can also be instance symmetries which are detectable in the instance data of this problem. Suppose there are two drones, drone1 and drone2, and the orders to be dispatched are x_1 and x_2 under the same arrival time and power. Then consider the following two situations:

```
Drone1 dispatch x1, drone2 dispatch x2
Drone1 dispatch x2, drone2 dispatch x1
```

The results of these two cases are the same, that is to say, the drone in the same state is symmetrical.

Therefore, this static symmetry breaking allows us to eliminate symmetric solutions through constraints. Lexicographic ordering constraints along one dimension of an array break the index symmetry of that dimension.

```
constraint forall(d1,d2 in DRONE where(d1<d2)/\
(arrival[d1]!=arrival[d2])/\
(charge[d1]!=charge[d2])/\
(order[d1]!=0)/\order[d2]!=0)
(lex_lesseq([order[d1]],[order[d2]]));
```

Or we can specify that the drone with smaller number send the order with smaller number:

```
constraint forall(d1,d2 in DRONE where(d1<d2)/\
(arrival[d1]!=arrival[d2])/\(charge[d1]!=charge[d2]/\(order[d1]!=0)/\order[d2]
!=0))
(order[d1]<order[d2]);
```

3 Compare

Running with gecode6.3.0, the results are summarized in the following table

	Symmetry is not excluded	Exclude only symmetry of resources	Only drone symmetry is excluded	The symmetry of resources and drone is excluded at the same time	Reason
drone0	1s 341msec	1s 261msec	1s 545msec	1s 103msec	
drone1	4s 370msec	6s 249msec	4s 600msec	1s 97msec	Resources are never idle
drone2	2s 156msec	1s 824msec	2s 89msec	1s 924msec	Drone is basically different
drone3	5m did not get the final result	5m did not get the final result	816msec	666msec	drone quantity equals order quantity
drone4	return Unsatisfied in 496msec	return Unsatisfied in 617msec	return Unsatisfied in 487msec	return Unsatisfied in 504msec	Too few resources
drone5	599msec	599msec	592msec	594msec	The drone status is different, and the resources are abundant
drone6	return Unsatisfied in 459msec	return Unsatisfied in 354msec	return Unsatisfied in 349msec	return Unsatisfied in 350msec	Too few resources to schedule orders
drone7	return Unsatisfied in 350msec	return Unsatisfied in 349msec	return Unsatisfied in 356msec	return Unsatisfied in 346msec	Not enough packs
drone8	return Unsatisfied in 379msec	return Unsatisfied in 385msec	return Unsatisfied in 384msec	return Unsatisfied in 375msec	Not enough packs
drone9	5m did not get the final result	4m 19s	1m 35s	1m 10s	The symmetry here is well reflected. Obviously, optimization plays a role

4 Sensitivity analysis

In this section, I will examine each drone file and consider which constraints have the greatest impact on profits.

	Stage G: modify the value
drone1	The profit obtained from the original data is 28, which is already the maximum profit. Without considering adding drones, we can consider modifying the pack time to [1,2,1,1] so that the time axis can be shortened.
drone2	The profit obtained from the original data is 44, which is already the maximum profit. Without considering adding drones, we can consider modifying the pack time to [1,2,1,1] so that the time axis can be shortened.
drone3	It is observed that the number of resources is still sufficient, but the drone lacks power and takes a long time to pack. The horizon is changed to 25, and the profit is increased from 54 to 59
drone4	Increase the number of resources and extend the time resources to [3,1,1,8,10,10]; Horizon is changed to 17; Profits can be withdrawn 59
drone5	The resources are abundant, and the drone does not need to be recharged. The profit is already the maximum and cannot be improved. Without considering adding drone, we can consider modifying the pack time to [1,2,3,1] so that the time axis can be shortened
drone6	Increase the number of resources = [1,1,1,2,3,3]; Get a profit of 32
drone7	Set pack resource to 2 and get a profit of 32
drone8	Set the pack resource to 2 and get a profit of 27, or extend the maximum time to 23 and get a profit of 29
drone9	Considering the power of the drone, the power of the third, fourth and seventh drones is changed to 10charge = [5,3,10,10,5,3,2,10,10,10], and the profit is increased from 83 to 98

In addition, the profits of drone 1, drone 2 and drone 5 cannot be increased. If drone drone1, drone2, and drone5 is added with an arrival time of 0 and a power of 10, the profits can be increased by 3, 5 and 5 respectively.

In summary, we can make the following decisions about the drone data:

Strategy 1: increase the number of resources. This is feasible for some inputs, such as drone3, drone6, drone7 and drone8. Due to the resource problem, the required solution was not obtained at the beginning.

Strategy 2: maintain high power for all drones. For example, drone1 sets the power of drone 3 to 10, so that it does not need to be charged quickly, and the profit will become 28.

Strategy 3: with the idea of strategy 2, the adjustment that can be made is to extend the maximum time limit. After the time is extended, all drone should not be charged quickly, and the cost of fast charging will be reduced.

Strategy 4: similar to the above, the weight of goods can be reduced to avoid charging, which may reduce the cost of fast charging.

Strategy 5: another way to increase the timeline is to reduce the packaging time and increase the timeline in disguise.

Strategy 6: make all orders ready in a very early time, so that all drones can return at the beginning. In this way, the sub problem becomes to select the most expensive order of ndrones.

5 Conclusion

Optimize drones symmetry: when the number of drone is far less than the order number and most of the drones are in the same state, the running speed of the program is significantly improved, and the optimal solution is quickly obtained. The number of drone is close to or equal to the order number and most of the dornes are in different states. The running speed has no impact. Due to an additional restriction, the solution will be slower.

When resources are abundant, the symmetry of resources is optimized, and the running speed of the program is significantly improved. When resources are scarce, solving the symmetry does not improve the efficiency.

6 Summary

We can break the detected symmetry so that we can spend less effort on solving, that is, to avoid multiple symmetric representations of the solution. When solving, ideally we should keep one member per symmetric class, because this may make the problem easier to solve.

7 Codes

```
enum ACTION = { LAND, INSPECT, FULLSERVICE, RECHARGE, PACK, TAKEOFF };
array[ACTION] of int: resources; % number of machines for each service

int: horizon;  % end time of planning horizon
set of int: TIME = 0..horizon;

int: norders;                    % number of orders;
set of int: ORDER = 1..norders;
array[ORDER] of int:  dist;      % distance warehouse to delivery
enum WC = { ULTRA, LIGHT, MEDIUM, HEAVY };
array[ORDER] of WC:  weight;     % weight catgeory of order
array[ORDER] of TIME: available; % when the order can be packed
array[ORDER] of int:  value;     % value of order
int: ndrones;
set of int: DRONE = 1..ndrones;
array[DRONE] of TIME: arrival;  % when they arrive back
array[DRONE] of int: charge;    % how much charge they have left
array[WC] of int: packtime;

/* Decisions */
array[DRONE, ACTION] of var TIME: start;                % start time for each action
array[DRONE, ACTION] of var 0..max(resources): resource; % resource used for each action
array[DRONE] of var 0..norders: order; % which order given to drone
array[DRONE] of var bool: fastcharge;

% StageA
% Each drone doesnt land before its arrival time
constraint forall(d in DRONE)(start[d,LAND] >= arrival[d]);
%StageA
% An order is not packed before it is available.
constraint forall(d in DRONE)(available[order[d]] <= start[d,PACK]);
% StageA
% The actions for each drone occur in the specified order after its arrival
% Recharge and pack operations need to be considered separately and written in stagec

constraint forall(d in DRONE)
```

```
constraint forall(d in DRONE)
 (start[d,LAND]<start[d,INSPECT]/\
                              start[d,INSPECT]<start[d,RECHARGE]
                             );

%StageA
%Each order is assigned to at most one drone.
%Means that each drone cannot have two identical orders
%There is a small problem here. There can be multiple drone with order=0

constraint forall(i,j in DRONE where((i!=j)/\(order[i]!=0)/\(order[j]!=0)))(order[i]!=order[j]);

%StageA Each drone and each action is assigned to a resource that actually exists in the warehouse (if
%the action is used) or 0 otherwise.

%StageA
% If the drone is given an order, then it must be assigned a resource for all actions, even if someof them may have 0 duration.
% If it has no order it only should be assigned a resource for LAND and INSPECT actions.

constraint forall(d in DRONE)(
                  if order[d] = 0
                  then resource[d,LAND] != 0 /\
                       resource[d,INSPECT] != 0 /\
                       resource[d,FULLSERVICE] = 0 /\
                       resource[d,RECHARGE]=0 /\
                       resource[d,PACK] = 0 /\
                       resource[d,TAKEOFF] = 0
                  else forall(a in ACTION)(
                         resource[d,a] != 0/\
                         start[d,TAKEOFF] <= horizon
                       )
                  endif
);

%StageB There is no overlap in actions assigned to the same resource
%Consider action start and end times
%Start time start time end (consider that the end time is equal to the start time of the next operation)
%When both drone1 and drone2 use one resource
%If (start1<end2 and start2<end1)
```

```
%Then the time axes coincide, and different resource IDS need to be allocated
constraint forall(a in { LAND, INSPECT, FULLSERVICE, RECHARGE, PACK })(
                  forall(i,j in DRONE
                        where (
                                (i!=j)/\
                                (start[i,a]<start[j,enum_next(ACTION,a)])/\
                                (start[j,a]<start[i,enum_next(ACTION,a)])
                         )
                  )
                  (resource[i,a]!=resource[j,a])
          );
constraint forall(i,j in DRONE where((i!=j)/\(start[i,TAKEOFF]==start[j,TAKEOFF])))(resource[i,TAKEOFF]!=resource[j,TAKEOFF]);

%If you want to use the same resource R, the next action of drone1 using the resource R first needs to be earlier than the current action of drone2 using the resource R later
constraint forall(a in { LAND, INSPECT, FULLSERVICE, RECHARGE, PACK })(
                  forall(i,j in DRONE
                        where(
                             (i!=j)/\
                             (resource[i,a]==resource[j,a])
                        )
                  )
                      (start[i,enum_next(ACTION,a)]>=start[j,a]\/
                       start[j,enum_next(ACTION,a)]>=start[i,a])
          );



%StageB There are never more actions of any type running than the number of resources for that task;
%Action resource limit: check whether the resource is within the limit at every time
%start<=t/\end>=t
%constraint forall(t in TIME)(
%                  forall(a in { LAND, INSPECT, FULLSERVICE, RECHARGE, PACK })(
 %                       sum(d in DRONE where (start[d,a] <= t)/\(start[d,enum_next(ACTION,a)] >= t))(1) <= resources[a]
  %                )
  %        );

%constraint forall(t in TIME)(sum(d in DRONE where start[d,TAKEOFF]=t)(1)<= resources[TAKEOFF]);

% StageC
% The packing time for the drone depends on the weight of the order it will carry, given by the data
constraint forall(d in DRONE)(start[d,PACK] + packtime[weight[order[d]]] = start[d,TAKEOFF]);

% StageC
%The charging time depends on the current charge level and the distance for the order it will
%carry. If the charge level is 10 then it doesnt need recharging, if the charge level is 5 or above
%and the distance at most 50km then it doesnt need recharging, otherwise it needs recharging.
%There are choices on the charging method: fastcharging takes 1 time unit, while slow charging
%takes 4 time units. We need to record the fastcharge decisions for each drone
%Forall (where) to judge whether to charge, current power, and order distance
%"If.." judge whether to charge quickly
%The current power, order distance and time are not enough,
%The time to operate the start packer. Push back 1. Add a true value to fast. Resource ID control
%"Else.." operation start packer time, push backward 4, resource ID control
constraint forall(d in DRONE)(
                  if (charge[d]==10)/\(order[d] != 0)
                  then (start[d,RECHARGE]=start[d,PACK]) /\
                       (resource[d,RECHARGE]!=0)
                  elseif (charge[d]>=5)/\(order[d] != 0)/\(dist[order[d]]<=50)
                  then  (start[d,RECHARGE]=start[d,PACK])/\
                        (resource[d,RECHARGE]!=0)%Do not charge
                  elseif(order[d] != 0)/\((start[d,RECHARGE]+packtime[weight[order[d]]]+4)>horizon)
                  %If the packing time and slow charging time exceed the maximum limit, use fast charging
                  then (start[d,RECHARGE]+1=start[d,PACK])/\
                        fastcharge[d]=true
                  else (start[d,RECHARGE]+4=start[d,PACK])
                  endif
);
% StageC
% A full service is required if the charge level is zero, and the next order is HEAVY or MEDIUM
%(so there is a next order); or if the inspection shows some problems. A full service requires
%10 time units. For now we assume the inspection always passes.
constraint forall(d in DRONE)(start[d,FULLSERVICE]=start[d,RECHARGE]);

% StageD
%Profit is given by the sum of the value of the orders that have taken
%off by the horizon time. The only cost we calculate is given by the number of fastcharges which
%each cost 5.
%Profit=sum (profit) -fast*5
%Time optimization = takeoff time drone time of all drone

var int: obj1;
constraint obj1 = sum(d in DRONE where(order[d] != 0))(value[order[d]])-sum(d in DRONE where(fastcharge[d]=true))(5);


% StageD
%The secondary objective is to minimize total turn around time, that is the total time
%spent before the drones take off with another order. The turnaround time for a drone is the take
%off time minus the arrival time. If a drone doesnt take an order its turn around time is 0.

var int: obj2;
constraint obj2 = sum(d in DRONE where(order[d] != 0))(start[d,TAKEOFF]-arrival[d]);

solve maximize (obj1*20 - obj2);

%Remove resource symmetry
%constraint forall(d in 1..ndrones-1)(lex_lesseq([ resource[d,a] | a in ACTION],[ resource[d+1,a] | a in ACTION]));
constraint forall (d1,d2 in DRONE,a in  { LAND, INSPECT, FULLSERVICE, RECHARGE, PACK }where(d1<d2)/\(start[d1,enum_next(ACTION,a)]=start[d2,enum_next(ACTION,a)])/\(start[d1,a]=start[d2,a]))(resource[d1,a]< resource[d2,a]);

%Remove the drone symmetry. This sentence is the same as the following two choices
%constraint forall(d1,d2 in DRONE where(d1<d2)/\(arrival[d1]!=arrival[d2])/\(charge[d1]!=charge[d2]/\(order[d1]!=0)/\order[d2]!=0))(order[d1]<order[d2]);

constraint forall(d1,d2 in DRONE where(d1<d2)/\(arrival[d1]!=arrival[d2])/\(charge[d1]!=charge[d2])/\(order[d1]!=0)/\order[d2]!=0)(lex_lesseq([ order[d1]],[ order[d2]]));
```