2023

RCSR Report

PROJECT 2

Pedro Domingos 59234
Iago Paulo 60198

Content

Problem Approach

The project description defines the situation of a logistics company that needs to spread among their available cars the parcels to deliver in a cost-efficient way. The company has stated some restrictions such as cars can only deliver in two zones if they share a common border, i.e., if they are contiguous and that obviously no cars can carry more total weight or volume than the limit established.

For the problem instances there is information on:

- Number of cars available.
- Zones that have common borders.
- Parcels.
- Weight and volume limits.

We initiated the problem-solving process by constructing a basic version of the solution. Subsequently, we iteratively refined the solution, progressively optimizing it until we arrived at our first delivered program, denoted as “parcel1.lp.” This program successfully addressed each sample scenario within a time frame of less than one second.

It is worth noting that in the optimizations chapter the changes showed are from a mix of different versions of the previous programs, meaning that the “before optimization” images might not be logically correlated. The optimization chapter not only discusses encoding improvements but also highlights optimizations derived from a deeper understanding of the problem.

After that we developed different versions of “parcel1.lp” by adding features described in group 2.

In our code we used consistently the same letters to make it more readable:

- C represents cars.
- L or Z represents locations or zones (they are the same logical element).
- P represents parcels (in this case the id of the parcel).
- T represents types or totals.
- N or X represents numbers.
- W represents weights.
- V represents volumes.

Optimizations

1º Optimization – car generation.

Our first optimization was changing the way that cars were being generated.

Before optimization:

```
% Getting cars
car(1..N) :- cars(N).
```

```
% Count assigned cars
usedCars(UC) :- UC = #count{ C : assign(_,C) }.
```

After optimization:

```
% Generate lastCar from 1 to N (number of cars available)
1{ lastCar(X) : X = 1..N }1 :- cars(N).

% Generate cars from 1 to lastCar
car(1..N) :- lastCar(N).

% The number of cars used is the same as the lastCar
usedCars(N) :- lastCar(N).
```

Before optimization we generated all cars and then we counted the number of cars that had any parcel assigned. After optimization we selected only a number between one and the total number of cars available and then generate every car from 1 to the chosen number, streamlining the generation process for increased efficiency. For instance, if we have 4 cars available, the second approach generates only 4 possible sets of solutions (only considering cars).

```
Solving...
Answer: 1
car(1)
Answer: 2
car(1) car(2)
Answer: 3
car(1) car(2) car(3) car(4)
Answer: 4
car(1) car(2) car(3)
SATISFIABLE

Models       : 4
Calls        : 1
Time         : 0.001s (Solving: 0.00s 1st Model: 0.00s Unsat: 0.00s)
CPU Time     : 0.000s
```

2º Optimization – zone assign.

Given the urban context of the problem within a metropolitan city, we have assumed that all zones are interconnected, signifying the absence of isolated zones, i.e., zones lacking common borders with any others specified in the problem instance. While acknowledging the strength of this assumption, it is noteworthy that none of the provided samples assesses scenarios and solving this new variant of the problem is easy as changing the zone assignment rule.

Before optimization:

```
% Creating zones
zone(Z) :- border(Z,_).
zone(Z) :- border(_,Z).

% Assinging zones to cars
0{ delivers(C,Z) : zone(Z) }2 :- car(C).
```

After optimization:

```
% Assign borders to cars
1{ deliversInBorder(C, Z1, Z2) : border(Z1,Z2) }1 :- car(C).

% Force the borders to be in ascending order
:- deliversInBorder(C1, Z1, Z2), deliversInBorder(C2, Z3, Z4), C1 < C2, Z1 > Z3.

% A car delivers in a zone if it delivers in a border of that zone
delivers(C,L) :- deliversInBorder(C,L,_).
delivers(C,L) :- deliversInBorder(C,_,L).
```

By associating a border with the car rather than the individual zone, we eliminate the cases where a car would be assigned to unconnected zones.

The pivotal aspect of this optimization step lies in the imposition of symmetry breakage by forcing the ascending order in the assigned borders. This specific measure prevents the generation of multiple correct solutions, thereby mitigating the potential escalation in running time. The emphasis here is on obtaining a singular correct solution, eliminating unnecessary computational overhead.

3º Optimization – parcel assign.

In our initial implementation, parcels were assigned to cars without specific constraints. In the subsequent iteration, the utilization of the "delivers" clause was introduced to eliminate instances where a particular car could not deliver a given parcel. Following the optimization phase, enhancements were made not only to streamline the number of elements in the rule body (car(C) was removed) but also to effectively break symmetries.

First implementation:

```
% Assigning parcels to cars
1{ assign(P,C) : car(C) }1 :- parcel(P,_,_,_).
```

Second implementation:

```
% A car delivers in a zone if it delivers in a border of that zone
delivers(C,L) :- car(C), deliversInBorder(C,L,_).
delivers(C,L) :- car(C), deliversInBorder(C,_,L).

% Assign parcels to cars
1{ assign(P,C) : delivers(C,L) }1 :- parcel(P,L,_,_).
```

After optimization:

```
% A car delivers in a zone if it delivers in a border of that zone
delivers(C,L) :- deliversInBorder(C,L,_).
delivers(C,L) :- deliversInBorder(C,_,L).

% Parcels ====================================

% Assign parcels to cars
1{ assign(P,C) : delivers(C,L) }1 :- parcel(P,L,_,_).

% Force the parcels to be in ascending order
:- assign(P1,C1), parcel(P1,L,_,_), assign(P2,C2), parcel(P2,L,_,_), P1 < P2, C1 > C2.
```

4º Optimization – weight and volume sum.

In our first implementation the predicates "sumWeight" and "sumVolume" were introduced to store the sum of the weight and volume of each car. After optimization, we simplified the solution by retaining only the two constraints, allowing them to implicitly handle the summation process.

Before optimization:

```
% Definitions ================================================================

% Get sum of parcel weights
sumWeight(C, SumW) :- car(C) , #sum { W,P : assign(P,C), parcel(P,_,_,W) } = SumW.

% Get sum of parcel volumes
sumVolume(C, SumV) :- car(C) , #sum { V,P : assign(P,C), parcel(P,L,T,W), parcelType(T,V) } = SumV.

% Restrictions ================================================================

% Cars can't exceed the weight limit.
:- car(C), weightLimit(LimitW), sumWeight(C, SumW), SumW > LimitW.

% Cars can't exceed the volume limit.
:- car(C), volumeLimit(LimitV), sumVolume(C, SumV), SumV > LimitV.
```

After optimization:

```
% Restrictions ================================================================

% Cars can't exceed the weight limit.
:- car(C), weightLimit(LimitW), #sum { W,P : assign(P,C), parcel(P,_,_,W) } > LimitW.

% Cars can't exceed the volume limit.
:- car(C), volumeLimit(LimitV), #sum { V,P : assign(P,C), parcel(P,_,T,_), parcelType(T,V) } > LimitV.
```

Optimization Results

After merging all the optimizations explained above, we created a comparison between the initial version and the optimized version with a maximum of 1 second per sample and here's the comparison result:

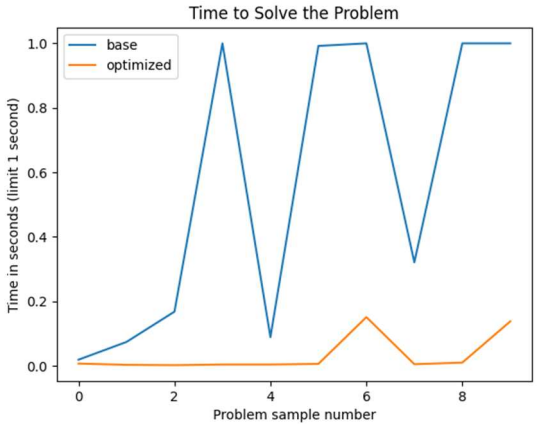

Figure 1 - initial version vs optimized version in terms of time to solve the problem.

It is worth noting that any sample requiring more than a second to solve would likely take more than a few minutes or even hours, emphasizing the significance of the optimization process, particularly the symmetry-breaking step.

First Group Program

```
% Generators ==================================================================
% Cars ========================================

% Generate lastCar from 1 to N (number of cars available)
1{ lastCar(X) : X = 1..N }1 :- cars(N).

% Generate cars from 1 to lastCar
car(1..N) :- lastCar(N).

% The number of cars used is the same as the lastCar
usedCars(N) :- lastCar(N).

% Zones ================================

% Assign borders to cars
1{ deliversInBorder(C, Z1, Z2) : border(Z1,Z2) }1 :- car(C).

% Force the borders to be in ascending order
:- deliversInBorder(C1, Z1, Z2), deliversInBorder(C2, Z3, Z4), C1 < C2, Z1 > Z3.

% A car delivers in a zone if it delivers in a border of that zone
delivers(C,Z) :- deliversInBorder(C,Z,_).
delivers(C,Z) :- deliversInBorder(C,_,Z).

% Parcels ==============================

% Assign parcels to cars
1{ assign(P,C) : delivers(C,Z) }1 :- parcel(P,Z,_,_).

% Force the parcels to be in ascending order
:- assign(P1,C1), parcel(P1,Z,_,_), assign(P2,C2), parcel(P2,Z,_,_), P1 < P2, C1 > C2.

% Restrictions ================================================================

% Cars can't exceed the weight limit.
:- car(C), weightLimit(LimitW), #sum { W,P : assign(P,C), parcel(P,_,_,W) } > LimitW.

% Cars can't exceed the volume limit.
:- car(C), volumeLimit(LimitV), #sum { V,P : assign(P,C), parcel(P,_,T,_), parcelType(T,V) } > LimitV.

% Optimization ================================================================

#minimize { C : usedCars(C) }.

#show assign/2.
#show usedCars/1.
```

*Figure 2 - "*parcel1*.lp" complete code.*

Group 2 Features

For the group 2 we had to decide which features implement from the following:

1. Change the restriction to delivering to different zones, so that cars may do deliveries in more than two zones as long as there is a delivery in each of the zones it has to pass through.
2. Attempt to reduce (in addition to reducing the number of overall used cars) the number of zones each of the cars has to deliver to, with the aim to minimize the number of time-consuming trips between zones.
3. Aim to balance (in addition to reducing the number of overall used cars) the number of deliveries per car to provide a fairer distribution of workload between the drivers.
4. You may provide an interesting presentation of the result(s) using clingraph.

We decided to implement the features 2, 3, 4 in files “parcel2g2.lp” and “parcel3g2.lp” and “viz.lp” respectively. We will detail the modifications implemented, as a substantial portion of the code remains unchanged.

Feature 2 – reducing zones visited.

The aim of this feature is to reduce the number of zones each car has to visit in order to minimize the time-consuming trips, for this, we calculated the total number of zones cars were visiting and minimized it.

Previously:

```
% Optimization ========================================================================

#minimize { C : usedCars(C) }.
```

After:

```
% Group 2 features ====================================================

% Counts the total number of distinct zones delivered to by all cars.
% Each car contributes to the count based on the unique zones it delivers to.
totalZonesDelivered(T) :- T = #count{ L,C : assign(P,C), parcel(P,L,_,_) }.

% Optimization ========================================================================

#minimize { C@2 : usedCars(C) }.
#minimize { T@1 : totalZonesDelivered(T) }.
```

Feature 3 – balance deliveries.

The aim of this feature is to balance the number of deliveries per car to provide a fairer distribution of workload between the drivers.

Previously:

```
% Optimization ===================================================================

#minimize { C : usedCars(C) }.
```

After:

```
% Group 2 features ================================================================

% Count the number of assignments per car
deliveriesPerCar(C, T) :- T = #count{ P,C : assign(P, C) }, car(C).

% Optimization ===================================================================

#minimize { C@2 : usedCars(C) }.
#minimize { D@1 : deliveriesPerCar(_, D) }.
```

Feature 4 – clingraph.

Procedure for generating clingraph presentations:

1. Ensure that the "Samples" folder is present within the project structure, containing all the relevant samples for analysis.
2. Execute the "run_all.sh" script. This script will autonomously establish necessary auxiliary folders and files, initializing the requisite data for graph creation. Subsequently, a new directory named "Graphs" will be generated within the project structure.
3. Within the newly created "Graphs" folder, there is a graph solution to each sample from the "Samples" folder.

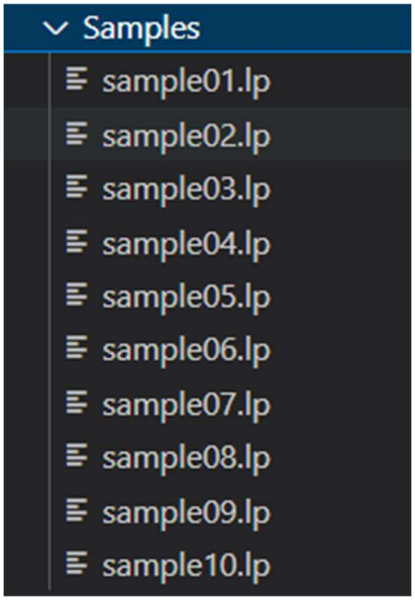

Figure 3 - Samples folder.

```
#!/bin/bash

# run_samples.sh
echo "Running run_samples.sh..."
./Scripts/run_samples.sh
echo "run_samples.sh completed."

# create_optimals.py
echo "Running create_optimals.py..."
python ./Scripts/create_optimals.py
echo "create_optimals.py completed."

# create_graphs.sh
echo "Running create_graphs.sh..."
./Scripts/create_graphs.sh
echo "create_graphs.sh completed."

echo "All scripts completed!"
```

Figure 4 - "run_all.sh" complete code.

“run_all.sh” executes the scripts:

- “run_samples.sh” – Creates a new folder named “Output_samples” containing the outputs from each sample.
- “create_optimals.py” – Does a text file manipulation to extract the optimal solution from the output text and stores it in the folder “Optimal_answers” as a lp file.
- “create_graphs.sh” – Executes the command to create the clingraph graph representation using the “viz.lp” and the “Optimal_answers” files, storing the new graph into the folder “Graphs”.

```
node((car, ID_CAR)) :- car(ID_CAR).
attr(node, (car, ID_CAR), style, filled) :- car(ID_CAR).
attr(node,(car, ID_CAR),label,
    @concat("🚗:",ID_CAR, " | 💪:", TW, "kg", " | 🔍:", TV, "dm³"))
    :- car(ID_CAR),
    TW = #sum { W,ID_PARCEL : assign(ID_PARCEL,ID_CAR),
    parcel(ID_PARCEL,_,_,W) },
    TV = #sum { V,ID_PARCEL : assign(ID_PARCEL,ID_CAR),
    parcel(ID_PARCEL,_,T,_), parcelType(T,V) } .

node((parcel, ID_PARCEL)) :- parcel(ID_PARCEL,_,_,_).
attr(node,(parcel, ID_PARCEL),label,
    @concat("📦_",ID_PARCEL," to ", TO_Z))
    :- parcel(ID_PARCEL,TO_Z,_,_).

node((border, Z1, Z2)) :- border(Z1,Z2).
attr(node,(border, Z1, Z2),label,
    @concat("📍(",Z1,"-",Z2,")"))
    :- border(Z1,Z2).


% Parcels --> Cars
edge(((parcel, ID_PARCEL) , (car, ID_CAR)))
    :- assign(ID_PARCEL, ID_CAR), parcel(ID_PARCEL,_,_,_).

% Cars --> Zones
edge(((car, ID_CAR) , (border, Z1, Z2)))
    :- deliversInBorder(ID_CAR, Z1, Z2).

% Graph label of limits
attr(graph, default, label,
    @concat("MAX💪:", MW, "kg", " | MAX🔍:",MV, "dm³"))
    :- weightLimit(MW), volumeLimit(MV).
```

Figure 5 - "viz.lp" complete code.

Test Samples Provided

During our project development we created a set of tests that evaluated the main restrictions of the problem and the group 2 implemented features. Here's a list of every sample and what do they test.

- "tsample01.lp" – Tests if the weight limit is respected.
- "tsample02.lp" – Tests if the volume limit is respected.
- "tsample03.lp" – Tests if a car can deliver in 2 neighboring zones.
- "tsample04.lp" – Tests if a car can't deliver in 2 non neighboring zones.
- "tsample05.lp" – Tests if a car can't deliver in 3 neighboring zones.
- "tsample06.lp" – Tests a mix of "tsample01.lp" and "tsample02.lp".
- "tsample07.lp" – Tests if feature 2 "parcel2g2.lp" works correctly.
- "tsample08.lp" – Tests if feature 3 "parcel3g2.lp" works correctly.
- "tsample09.lp" – Tests both features 2 and 3.

For every sample test file there is a complete description of what does the test evaluate and what is the format of the correct answer, here's an example:

```
% This test checks if any of the features 2 and 3 from group 2 works correctly.

border(a,b).

parcelType(1,1).

weightLimit(400).
volumeLimit(1000).

cars(3).

parcel(1,a,1,100).
parcel(2,a,1,100).
parcel(3,a,1,100).
parcel(4,a,1,100).
parcel(5,a,1,100).
parcel(6,b,1,100).
parcel(7,b,1,100).
parcel(8,b,1,100).
parcel(9,b,1,100).

% Answer should have 3 cars independently of the feature:

% For the feature which reduces the number of total zones delivered (file "parcel2g2.lp"):
% 2 cars for zone A since no car can carry 5 parcels
% 1 car for zone B delivering the 4 parcels

% For the feature which aims to balance the number of delivers per car (file "parcel3g2.lp"):
% 3 cars delivering 3 parcels each
% This balances the delivers per car to 3
```

Figure 6 - "tsample09.lp" complete code.