Automated Reasoning Project

Stefano Monte

March 2024

Contents

1 Introduction

The project aims to evaluate the performance of Minizinc and Answer Set Programming (ASP) by analyzing their capabilities in solving a specific problem. This involves developing programs in both Minizinc and ASP to address the stated problem, considering the provided constraints to generate solutions effectively.

Subsequently, a comprehensive set of 12 benchmark instances will be crafted to assess the performance of the implementations. These instances will be generated automatically to cover a diverse range of difficulty levels, ensuring representation of easy, average, and hard scenarios. The intention is to create instances that can be solved within varying timeframes, from a couple of seconds to a few minutes, with some instances potentially exceeding a designated timeout limit.

Each benchmark instance will undergo testing using both Minizinc and ASP encodings, with exploration of different search strategies. To manage execution time and prevent excessively long runs, a timeout of 5 minutes will be enforced for each test.

2 Assignment

2.1 Text of the problem

A cyclist wants to explore their area by visiting all the roads within it (the road network is given as input, with intersections as nodes and each road segment as an edge with its length provided). Fortunately, the area is all flat terrain. The only constraint is that they can cover at most k (input parameter, for example, 100) kilometers per day. The cyclist doesn't necessarily have to start from the same node every day (you can imagine they reach certain areas by car before unloading the bike and starting to pedal), but they must return exactly to the node they started from each day. Repeating the same road is not prohibited, whether on the same day or on different days. However, the objective is to eventually traverse all the roads by bike. In the solution, minimize the number of days required. Once a solution is found, try increasing the limit of k slightly until you gain an extra day.

2.2 Constraints

Upon reviewing the assignment, it is feasible to extract the essential constraints required to tackle the problem at hand:

- The cyclist can cover at most k (given) kilometers per day.
- The cyclist must return to the same starting node each day.
- The cyclist can start from any node each day but he must return to the same starting node at the end of the trip.
- There are no restrictions on repeating the same road, whether on the same day or on different days.
- The objective is to visit all the roads by bike.

2.3 Function to minimize

Given that this is a Constraint Optimization Problem, the objective is to minimize a certain function. In this context, the function to minimize represents the number of days required for the cyclist to visit all the roads within the specified area.

3 Minizinc model

The Minizinc model is designed to take input parameters such as the number of nodes, the maximum distance the cyclist can ride each day, and the graph representing the streets of the area. The area is depicted as a matrix where each tuple of indices represents intersections connected by a road. Each cell in the matrix indicates the length of the road between two intersections, with a value of zero indicating the absence of a road.

The daily itinerary is represented by a matrix where each row corresponds to a different day and contains the intersections visited on that specific day. From these intersections, it is straightforward to trace the corresponding roads.

To minimize the number of days required, the path matrix should ideally contain as many rows with all zeros as possible, indicating that the cyclist has completed exploration of all roads.

3.1 Inputs

```
int: intersections;

int: max_km_per_day;

array [1..intersections, 1..intersections] of int: distance;
```

- **intersections**: Defines the number of nodes (intersections) in the road network.
- **max_km_per_day**: Specifies the maximum distance that can be traveled in a single day.
- **distance**: Represents the distances between pairs of nodes in the road network.

3.2 Variables

```
int: roads = (intersections * 2) - 1;

int: max_days = sum(i in 1..intersections, j in 1..intersections)(if distance[i,j] != 0 then 1 else 0 endif) div 2;

array [1..max_days, 1..roads] of var 0..intersections: path;

var 1..max_days: days;
```

- **roads**: Calculates the total number of roads in the network, which is set to (intersections * 2) - 1 in order to have enough moves.
- **max_days**: Determines the maximum number of days required to traverse all roads. It's calculated based on the number of roads.
- **path**: An array representing the cyclist's path for each day. Each element of the array corresponds to a road segment between two nodes. The value indicates the node visited or 0 if the road is not traversed.
- **days**: Represents the total number of necessary days to traverse all roads.

3.3 Predicates

```
predicate backToStart(var int: d) =
  exists(j in 2..roads - 1)(path[d, 1] == path[d,j] /\ forall(k in j+1..roads)(path[d,k] == 0));

predicate pathExistance(var int: d) =
  forall(j in 1..roads - 1)(if path[d,j] != 0 /\ path[d,j+1] != 0
    then path[d,j] != path[d,j+1] /\ distance[path[d,j], path[d,j+1]] != 0 endif);

predicate inMaxKm(var int: d) =
  sum(j in 1..roads - 1 where path[d, j+1] != 0)(distance[path[d,j], path[d,j+1]]) <= max_km_per_day;

predicate exploreAllRoads() =
  forall(i in 1..intersections, j in 1..intersections where i != j /\ distance[i,j] != 0)(
    exists(k in 1..max_days, l in 1..roads - 1)(
      (path[k,l] == i /\ path[k,l+1] == j) \/
      (path[k,l] == j /\ path[k,l+1] == i)
    )
  );
```

- **backToStart**: Ensures that the cyclist returns to the starting node at the end of each day.
- **pathExistance**: Verifies that a valid path exists between consecutive nodes in each day's path.
- **inMaxKM**: Checks that the total distance traveled in a day does not exceed the maximum allowed distance.
- **exploreAllRoads**: Confirms that all roads are traversed at least once in the solution.

3.4 Constraints

```
constraint forall(d in 1..max_days)(
  backToStart(d) /\ pathExistance(d) /\ inMaxKm(d));

constraint exploreAllRoads();

constraint days = max(i in 1..max_days)(if path[i,1] != 0 then i else 0 endif);
```

- **Main constraint**: Combines the predicates to enforce constraints on the cyclist's path for each day.
- **exploreAllRoads**: Ensures that all roads are traversed at least once in the solution. It verifies that the namesake predicate is verified.
- **Days Calculation**: Determines the number of days needed to traverse all roads. It's calculated based on the last row where the values are not 0 in the path array.

3.5 Minimization and Strategies

```
solve :: int_search(path, first_fail, indomain_max)
minimize days;
```

Since the goal is to find a the minimum amount of days necessary to explore all the road we have to minimize the variable days which represent the counter of the days the cyclist need. After several experiments, the search strategy that yielded the best results was to guide the search to first consider the variable with the smallest domain, and for that variable, to consider the maximum value in its domain first.

3.6 Example of solution

```
path =
[| 3, 4, 1, 2, 3, 0, 0
 | 3, 4, 2, 4, 1, 3, 0
 | 0, 0, 0, 0, 0, 0, 0
 | 0, 0, 0, 0, 0, 0, 0
 | 0, 0, 0, 0, 0, 0, 0
 | 0, 0, 0, 0, 0, 0, 0
 |];
days = 2;
_objective = 2;
```

This matrix represents the solutions, where each row represents the cyclist's trip for each day. When the cyclist completes their trip, the remaining nodes are set to zero.

4 ASP model

The ASP model consists of facts, which serve as the input for the problem, as well as rules and constraints. The model tracks the number of roads visited each day and ensures that all constraints are satisfied in order to find the correct solution.

4.1 Facts

Firstly, the program includes the facts of the problem, which consist of the inputs provided to solve the problem. These inputs typically encompass the number of days allocated for the trip, the maximum number of roads that can be traveled per day, the cyclist's autonomy (maximum distance the cyclist can travel per day), and the graph representing the streets.

- **day(1..i)**: Represents the days available for the cyclist to travel.
- **roadsTraveledPerDay(1..j)**: Specifies the number of roads the cyclist can travel per day.
- **max_km_per_day**: Constant representing the maximum distance the cyclist can travel in a day.
- **road(I1, I2, Length)**: Describes the roads in the network, with their respective lengths.

4.2 Rules

```
1{requiredDaysToVisitAllRoads(D) : day(D)}1.

1{travel(D, R, I1, I2) : road(I1, I2, _)}1 :- roadsTraveledPerDay(R), day(D), roadVisitedPerDay(D, V), R <= V.

1{roadVisitedPerDay(D,R) : roadsTraveledPerDay(R)}1 :- requiredDaysToVisitAllRoads(ND), day(D), D <= ND.

visited(I1, I2) :- travel(_, _, I1, I2).

kmTravelledPerDay(KM, D) :- day(D), requiredDaysToVisitAllRoads(ND), D <= ND,
                            KM = #sum{Length, R : travel(D, R, A, B), road(A, B, Length)}.
```

- **requiredDaysToVisitAllRoads(D)**: Indicates the number of days required for the cyclist to visit all roads.
- **travel(D, R, I1, I2)**: Specifies the roads chosen by the cyclist to travel on a particular day (on the same day and move the cyclcist can take only a single road).
- **roadVisitedPerDay(D, R)**: Specifies the number of roads visited by the cyclist on each day.
- **visited(I1, I2)**: Indicates if a road has been visited by the cyclist (if there is a trip with the specified road in it).
- **kmTravelledPerDay(KM, D)**: Calculates the total distance traveled by the cyclist on each day.

4.3 Constraints

```
:- day(D), travel(D, 1, S, _), roadVisitedPerDay(D, R), travel(D, R, _, F), S != F.

:- travel(D, R, _, F), travel(D, R+1, S, _), F != S.

:- day(D), kmTravelledPerDay(KM, D), KM > max_km_per_day.

:- not visited(I1, I2), not visited(I2, I1), road(I1, I2, _).
```

- **Starting and Ending Node**: Ensures that the cyclist starts and ends their travel at the same node each day.
- **Connected Travel**: Ensures that the roads traveled by the cyclist on a particular day are connected.
- **Maximum Distance**: Enforces the constraint that the cyclist cannot travel more than the maximum allowed distance per day.
- **All Roads Visited**: Ensures that all roads in the network are visited by the cyclist.

4.4 Minimization

```
#minimize {D:requiredDaysToVisitAllRoads(D)}.

#show travel/4.
#show requiredDaysToVisitAllRoads/1.
```

The minimize directive is utilized to minimize the number of days required to visit all roads within the problem. Additionally, the show directives are employed to exhibit the travel plans and the number of necessary days to visit all roads in the program's output.

4.5 Example of solution

```
Solving...
Answer: 1
requiredDaysToVisitAllRoads(2) travel(1,1,4,1) travel(1,2,1,4) travel(1,3,4,2) travel(1,4,2,3) travel(1,5,3,2) travel(1,6,2,4)
travel(2,1,2,1) travel(2,2,1,4) travel(2,3,4,1) travel(2,4,1,4) travel(2,5,4,3) travel(2,6,3,1) travel(2,7,1,2)
Optimization: 2
OPTIMUM FOUND
```

The output of the program displays the number of days required by the cyclist to complete the exploration, along with the itinerary for each day of the trip.

5 Benchmarks

We will assess the performance of two programs using a set of 12 test cases. These cases are categorized into 4 easy, 4 medium, and 4 hard instances. Each program will have a time limit of 5 minutes (300 seconds) to solve each test case.

The Minizinc program underwent testing with three different solvers: Gecode 6.3.0, Chuffed 0.13.1, and CP-SAT 9.8.3296 comparing time consumption with and without heuristics. On the other hand, the ASP program was evaluated using the Clingo solver and no heuristics were used.

5.1 Instances

The test cases are automatically generated using a Python test generator, which creates a random map consisting of roads and intersections. Each road is assigned a random distance. The only constraint applied is for the maximum distance the cyclist can ride each day. Since the cyclist must return to the starting node each day, the maximum length of any road is at most half the cyclist's daily distance autonomy. Additionally, the cyclist's daily distance is randomly generated within the range of 50 to 100 kilometers.

Instance	**Intersections**	**Roads**	**Max km per day**
easy 1	4	5	96
easy 2	4	6	88
easy 3	5	6	76
easy 4	5	7	95
medium 1	6	7	65
medium 2	6	8	51
medium 3	6	9	96
medium 4	7	8	82
hard 1	8	10	77
hard 2	9	11	97
hard 3	10	11	50
hard 4	11	13	83

5.2 Results without euristics

Instance	**Days**	**ASP (s)**	**Minizinc (s)**		
		Clingo	Gecode	Chuffed	CP-SAT
Easy 1	2	0.391	0.447	0.366	0.829
Easy 2	2	0.406	0.485	0.384	1.297
Easy 3	1	0.273	0.460	0.401	1.203
Easy 4	2	1.850	0.569	0.494	2.954
Medium 1	4	8.849	142.287	0.750	2.631
Medium 2	4	13.023	131.469	1.189	8.207
Medium 3	5	84.274	> 300	3.886	15.001
Medium 4	4	57.971	> 300	57.608	76.064
Hard 1	4	> 300	> 300	89.418	287.621
Hard 2	-	> 300	> 300	> 300	> 300
Hard 3	-	> 300	> 300	> 300	> 300
Hard 4	-	> 300	> 300	> 300	> 300

Based on the results, it's evident that Minizinc, particularly when using the Chuffed 0.13.1 solving configuration, outperformed the other two solving configurations. Gecode was observed to be the least optimal. Comparing ASP and Minizinc using Chuffed, Minizinc consistently exhibited faster performance across nearly all resolutions. While the difference was minimal in the easy instances, it became more pronounced in the medium instances. Notably, only two solving configurations of Minizinc were able to solve the first hard problem within the designated 5-minute timeframe, while ASP and Gecode configurations failed to do so.

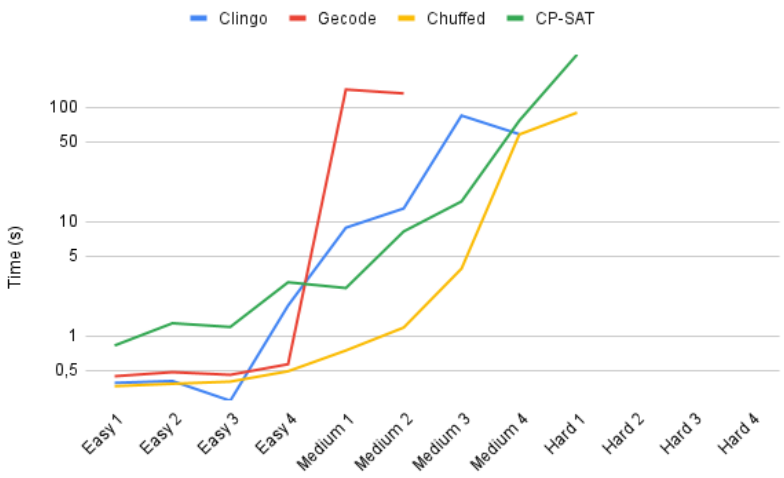

Figure 1: Time comparison graph of all the solvers in logarithmic scale.

5.3 Minizinc results with heuristics

Instance	**Days**	**Minizinc (s)**		
		Gecode	Chuffed	CP-SAT
Easy 1	2	1.610	0.366	0.896
Easy 2	2	0.516	0.418	0.793
Easy 3	1	0.528	0.395	0.998
Easy 4	2	1.495	0.578	1.278
Medium 1	4	> 300	0.878	1.773
Medium 2	4	> 300	1.998	2.575
Medium 3	5	> 300	5.338	3.397
Medium 4	4	> 300	2.099	4.594
Hard 1	4	> 300	37.020	20.594
Hard 2	7	> 300	22.059	> 300
Hard 3	7	> 300	83.236	> 300
Hard 4	-	> 300	> 300	> 300

The heuristics applied to the Minizinc program using the Chuffed and CP-SAT solvers have resulted in slightly slower times for easier-to-solve problems but have significantly improved efficiency for more difficult ones. In fact, these two solvers managed to solve even the second hard problem within the 5-minute time limit. On the contrary, noticeable deteriorations were observed when using Gecode. Not only did it achieve worse timings for easier problems, but it also failed to solve problems of moderate difficulty, which it had managed to solve without the heuristics.

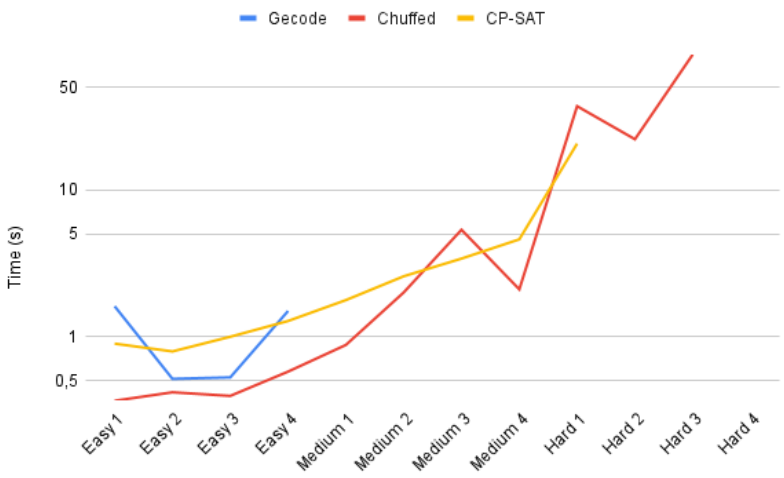

Figure 2: Time comparison graph of the 3 Minizinc solvers in logarithmic scale.

5.4 Comparison of Minizinc solvers with and without heuristics

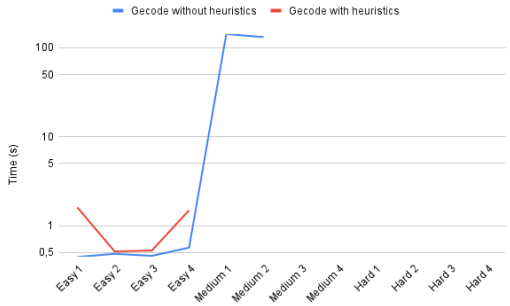

Figure 3: Gecode graph in logarithmic scale.

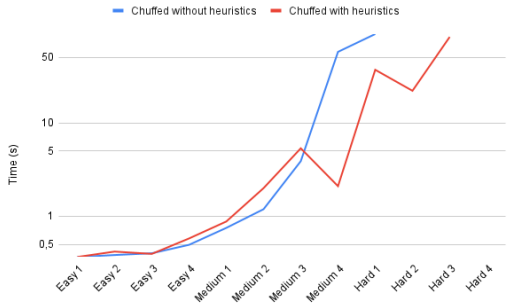

Figure 4: Chuffed graph in logarithmic scale.

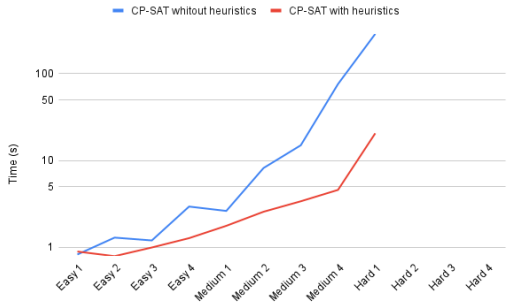

Figure 5: CP-SAT graph in logarithmic scale.

The comparison between Chuffed, CP-SAT, and Gecode highlights the variation in solver performance based on the heuristic used. While Chuffed and CP-SAT showed significant improvements, Gecode disappointed. This demonstrates that heuristics are solver-specific, and what works for one may not work as well for another. Careful selection and optimization of heuristics based on problem characteristics and solver architecture are crucial. This requires a thorough evaluation of different heuristic strategies to maximize solver efficiency across various problem domains.

5.5 Gaining a day

Following the time comparison, let's now analyze the additional kilometers needed to the previous limit in order to gain an extra day in exploration.

Instance	**Before (km)**	**After (km)**	**Gap (km)**
Easy 1	96	119	23
Easy 2	88	110	22
Easy 3	76	76	0
Easy 4	95	164	69
Medium 1	65	91	26
Medium 2	51	60	9
Medium 3	96	98	2
Medium 4	82	96	14
Hard 1	77	82	5
Hard 2	97	-	-
Hard 3	50	-	-
Hard 4	83	-	-

From the results, it is evident that for easier problems, significantly more kilometers need to be added in order to gain an additional day, whereas for more difficult problems, fewer kilometers suffice. One possible explanation for this occurrence is attributed to the larger variety of street choices available in larger areas. In simpler problems with fewer streets, the cyclist may need to cover more distance to uncover new routes and complete the journey within the same timeframe. Conversely, in larger areas with more streets, the cyclist has a wider selection of routes to explore, potentially requiring less distance to discover new paths and complete the journey.