Autonomous Drone Path Planning using Answer Set Programming

Leonardo Pohl

University of Potsdam

Abstract. Answer Set Programming is used in many different fields for various tasks from natural language processing and RNA sequencing to composing music. In this paper, we describe how this method can be applied to support an autonomous drone by finding a path while reducing the height difference to decrease the travel time and fuel consumption. Besides, a main focus of the present paper is to show how data from the real world can be abstracted to be used as facts for an ASP program.

Keywords: Autonomous Path Finding · Answer Set Programming · Data Representation

1 Introduction

Last year we thought of a drone, which should be able to carry a payload autonomously from a start point to an end point and back. While designing the drone, we were also thinking of ways to find a path to the end point.

We decided to use Answer Set Programming since its feature set seemed to fit the project's requirements and we were intersted in seeing how to implement a solution using ASP in connection with other systems.

In this paper, we will take a glance at the path finding ASP program and the data gathering tool, which is used to create facts that include the start and end point and the local height data.

If the reader is not familiar with Answer Set Programming or interested in learning more about it, we would recommend reading the papers [1] and [2], but we will try to explain the code snippets understandably.

2 Implementation

We decided to use Answer Set Programming to find a path leading from the drone's start position to its destination. To gather the necessary data and finally visualize the path we used Python and the Google Maps Elevation API. The Implementation can be found on GitHub[1].

[1] `https://github.com/LeonardoPohl/asp-dronepath-paper`

2.1 Data Gathering

The most important step in gathering the data is to decide what kind of data we need, how it is distributed and how we can reformat it so that it can be used as a fact, a way to represent knowledge so it can be used by an ASP-program, for the path finding program.

Because there is almost[2] an infinite number of points between any two points in space we need to find a way of distributing a number of points over an area between the start and end point. hese points should not be located on a line because we might want to find a path around high terrain like mountains. Moreover, this area should be applicable all over the world, and it should be the same shape no matter how the points are oriented. Additionally, we do not need that many points at the start and end point because they are not likely to be used.

Another requirement for the grid is a simple mapping between the geo-coordinates and the relative coordinates because ASP is constrained to integer valued input and the rules are simpler to define in a local system.

A Rectanglular Grid - The first idea was to create a rectangle which would have both points at opposing ends. This could be extended to a version which scales with the distance to both points because the start and landing would need the most information to create a smooth take off and landing. The problem with this solution is that it is not unambiguous.

(a) Regular Height Grid (b) Scaled Height Grid

Fig. 1: First version of the grid between Potsdam Central Station and Potsdam Griebnitzsee

But this rotation, which had to be made to create this rectangle, can not be defined clearly, and we would have problems when converting these points to relative points.

[2] In theory there is an infinite amount of different points but in the real world we have physical limitations.

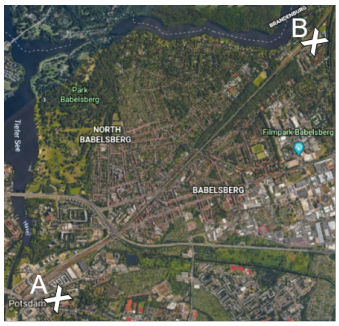
A Diamond-Shaped Grid - We decided to use a similar but slightly different unambiguous approach which, in contrast to the first approach, would not create a rectangle but a diamond. The diamond is defined by an angle alpha and the geo-coordinates are divided by a value accuracy. With the value accuracy, we can generate integer values varying in granularity.

First we create a vector between the start and end point (Vector $\overrightarrow{AB}$ with a length d_{A-B} in figure 2b). Then we calculate the maximum width of the parallelogram (r_{max} in figure 2b) for each step using the *arctan* of alpha times the iterative distance. An iteration over the width of the diamond gives rise to new grid points located within area of interest.

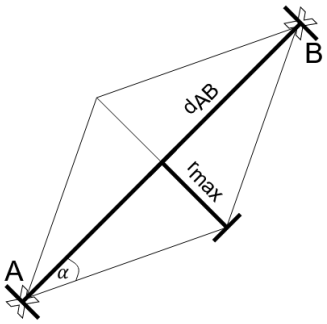
With that we can derive the following algorithm:

```
1|def diamond_grid(pt_A, pt_B, alpha, accuracy):
2|  vec_AB = vector(pt_A, pt_B)
3|  points = []
4|
5|  for d in range(0, length(vec_AB)):
7|    if d less_than_halfway:
8|        r_max_d = tan(alpha) * k
9|    else:
10|       r_max_d = tan(alpha) * (length(vec_AB) - d)
11|
12|   for r in range(-r_max_d, r_max_d):
13|       beta = arctan(r / d)
14|       rotate_vec(pt_A, pt_B, beta)
15|
16|       length_AC = sqrt(d^2 + r^2)
17|       extend_vec(pt_A, pt_B, length_AC / d)
18|
19|       point_C = vec_AB + pt_A
20|       points.append(point_C.extend(1 / accuracy))
21|
22|  return points
```

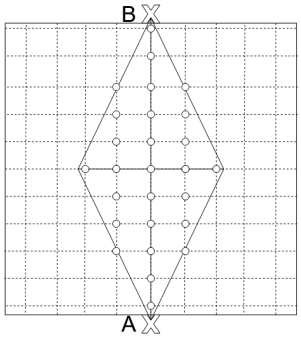
First we define a vector between point A and point B. Then we iterate over every point between A and B, calculating the maximum distance, r_max_d, to the vector vec_AB using the distance d to either A or B depending on which is the closest in line 7 through 10 using alpha. Afterwards we iterate over every point which is between r_max_d and -r_max_d. In this iteration we calculate a new point by rotating the vector vec_AB. Line 16 and 17 extend the vector to be of the length of the hypotenuse.



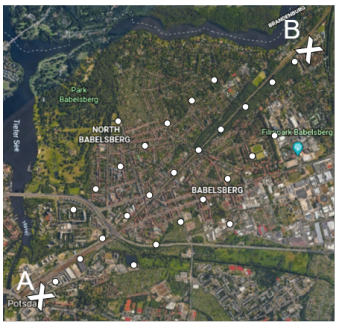
(a) Map with a start and end point



(b) Abstraction of the map to generate constraints



(c) Mapping the constraints to a grid to generate points



(d) Resulting points shown on a map

Fig. 2: Updated version of the grid. A Diamond spanned between point A and B

As a result of this iterative procedure, we obtain a list of points which have both relative and real coordinates. With these coordinates we create a query for Googles Elevation API and process the request. To save API calls, we keep the result in separate JSON files, which are labelled according to thier parameters for further usage.

2.2 Path Finding

After defining the start and the end point, the next step is the creation of the ASP facts with the relative coordinates and the height in meters. The created facts take the form:

```
point(X,Y,H).
start(X,Y,H).
end(X,Y,H).
```

Path Creation - The first step is to generate a valid path applying the following rules:

```
1 |  isPoint(point(X, Y, Z)) :- point(X, Y, Z).
2 |  {path(A, B)} :- isPoint(A), isPoint(B).
3 |  :- path(point(X1, Y1, _), point(X2, Y2, _)),
  |     |X1 - X2| + |Y1 - Y2| <> 1.
4 |  :- path(point(X1, Y1, _), point(X2, Y2, _)),
  |     X2 - X1 < 0.
5 |  :- path(point(X1, Y1, _), point(X2, Y2, _)),
  |     path(point(X2, Y2, _), point(X1, Y1, _)).
6 |  :- not path(A, _), start(A).
7 |  :- not path(_, B), end(B).
  |
8 |  connected(A) :- start(A).
9 |  connected(B) :- connected(A), path(A, B).
10|  :- path(A, B), not connected(B), not connected(A).
```

Line 1 simplifies line 2, which generates path segments leading from any point A to another point B. In line 3, we define that the drone can only move to a neighbouring point. Line 5 improves the efficiency by ensuring that no path can be undone.

Line 6 limits the movement of the drone to only move forward or sideways, this reduces the number of ground rules to improve the solving time. Line 7 and 8 state that there has to be a path segment starting at the start point, and another section has to end at the end point.

To ensure that the path just leads from the start to the end, line 8 through 10 recursively ensure that every point on a path is connected to the start. The connection to the end point is ensured by line 7.

Minimization - To find an optimal path, we need to reduce the height difference between every connected pair of points and the number of points in a path.

```
1|  #minimize {|H1–H2| + 1, X1, Y1, X2, Y2 :
 |              path(point(X1, Y1, H1), point(X2, Y2, H2))}.
```

This rule minimizes the height difference with $|H1 - H2|$ while also reducing the number of paths adding 1. Thus, we get a path which not only minimizes the amount of paths but also minimizes the height difference between any two points. Consequently creating a path which is supposed to be the most efficient.

2.3 Data Evaluation

For now, the user has to copy the optimized model and paste it into an output file so the program can parse it. In the evaluation part of the program, the grid

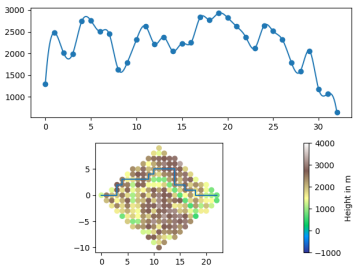
is recreated, and then the output of the Path Finding program is mapped to the grid. This is possible because of the unambiguity of the grid.

To see whether or not the path is realistic it is drawn on a height map of the relative grid and then plotted on an actual map to see if the mapping was successful.

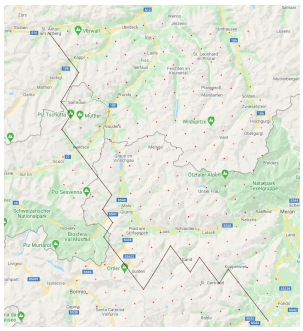
3 Result

To test the performance of the program with respect to path finding in a area with large height deviations, we chose a route between St. Anton (Germany) and Malgolo (Italy).

We chose an angle of 40° and an accuracy of 20. These parameters created 236 Points, and it took the clingo 1179s to find the optimal path using eight threads.



(a) Path Drawn on Relative Points (The curve is not an actual relief it is just there to make it look neat)



(b) Path Plotted on Map

Fig. 3: Resulting path from St. Anton (Germany) to Malgolo (Italy) (a) using relative points and (b) plotted on the actual map.

The path seems to be optimal regarding the actual height difference but compared to real-life it is not. In real-life this path does not seem to be optimal because in reality to minimize the height difference one would usually go around the mountain and not over it. and thus, it is only minimal for the given numerical setup but not possible to estimate whether it is also the optimal situation for the real world.

If we were to use this data for an actual flight we would need to query the height of the whole path to see if any obstacles are on the route and try to fly over or around these. It might be an option to recalculate the path further along the way to optimize the course on the fly.

4 Future Work

The main focus if we were to further improve the software would be to focus on usability. At the time of writing, the user has to restart the program after it created the points for the first time. Then the user needs to start clingo, wait for it to finish, find the optimal solution and copy the model. These steps should happen automatically when starting the program. Yet for a proof of concept, the current state is sufficient.

Furthermore, we should find a way of checking how many points are queried, and we should be setting better limits, which prevent the user from starting a task which is unsolvable in an average human lifespan.

In combination with that, we should improve the efficiency of the ASP program since this is still a severe bottleneck. We will not be able to eliminate it entirely, but we should strive for less solving time.

Moreover, this program uses areas where it is prohibited to fly a drone, this feature should be added because the drone really should not fly through these. Still, with that, we would need to think about the possibility that there is no path and maybe we need to change the angle or accuracy adaptively.

5 Conclusion

In this paper we showed that it is possible to dynamically create grid which can be transformed to ASP facts to find a path which is optimal not only regarding the distance but also regarding the elevation.

In conclusion, we can say that for a proof of concept the program suffices, but if this is ever going to be used, we need to improve the user experience, and the program requires further testing and debugging to improve the stability. But we were able to show that answer set programming is an excellent way of solving this kind of problem. It would also be really interesting to see how the accuracy and different weights for the minimization functions might influence the path to optimize the solving process.

In summary we were able to show that an implementation with answer set programming is not only possible but also feasible.

References

1. Vladimir Lifschitz.: Answer Sets and the Language of Answer Set Programming. AI Magazine **37**(3), 7–12 (2016)
2. Tomi Janhunen, Ilkka Niemelä.: The Answer Set Programming Paradigm. AI Magazine **37**(3), 13–24 (2016)