Planning Project - A Good Snowman is Hard to Plan

Matteo Paparo - Mat 264387

Introduction

The following paper consists of a study regarding the use of *PDDL Numeric*; specifically, the "A Good Snowman is Hard to Plan" problem was addressed. The work is centered on the creation of snowmen, where several gradual conversions were made to the original domain. The choice was made to proceed by converting the most important components of the domain. The objective of this analysis is to verify the impact of using PDDL Numeric on the proposed problem.

Domain

The domain is represented on an $n \times m$ grid (not necessarily regular) where each cell can be covered in snow, and depending on the problem, there will be 3 to 6 snowballs for creating the snowmen. With reference to the problem's objective, 3 different domains were identified, for creating 1 and 2 snowmen, respectively.
The rules to be followed are:

- If a snowball passes over a snowy cell, and the snowball is not at its maximum size, it will increase in size;
- It is possible to create a stack of snowballs if the ball on top is smaller than the ball below it;
- To move a snowball, the player must be in a cell adjacent to the snowball. If the destination cell is free, or contains a larger snowball, then the snowball can be moved, and the player will move to the cell previously occupied by the ball.

Types, objects and fluents
The objects considered are the following: *loc, dir*, and *ball*. The *loc* type represents the position of a grid cell. The *dir* type enumerates the four directions in which the character and the balls can move (up, down, left, right), and the *ball* type defines all the balls present in the scenario.

To model the current state, in the basic domain, the following predicates were defined:

- (snow ?l - loc): location l is covered in snow;
- (next ?from ?to - loc ?d - dir): locations from and to are adjacent; location to is in direction d relative to from;
- (occ ?l - loc): location l contains at least one ball;
- (char at ?l - loc): character is at location l;
- (ball at ?b - ball ?l - loc): ball b is at location l;
- (ball size s ?b - ball): ball b has small size;
- (ball size m ?b - ball): ball b has medium size;
- (ball size l ?b - ball): ball b has large size;
- (goal): the goal is satisfied.

Actions
The actions considered are the following: *move_ character*, which, as the name suggests, allows the character to move to an adjacent cell, *move_ ball*, which allows a snowball to be moved to an adjacent cell, and *goal*, which allows checking if the goal has been reached.

With reference to the original version, the following results were obtained[1]:

Domain	Problems	Solved
1	33	25
2	13	1

Conversion to PDDL Numeric

The adaptation of the domain to PDDL Numeric was carried out gradually. Starting from the base domain, the most functional conversion was implemented, which was the conversion of the predicates concerning ball_size into a numeric function. Specifically, to identify the three sizes (small, medium, and large) of the snowballs, three different values were assigned: 1 for small, 2 for medium, and 3 for large. Subsequently, an adaptation of the grid definition methodology was carried out, converting the relative predicate into two numeric functions to identify the cell with coordinates. To conclude, a conversion with a different verification methodology for the creation of snowmen.

In summary, we identify 3 different versions:

1 **Goal**: which includes the conversion of predicates regarding the size of the snowballs and the goal;

2 **Coord**: which, in addition to the previous conversion, identifies locations with numeric coordinates;

3 **Count_ball**: which, in addition to the first version, keeps track of the number of snowballs in each location.

Goal - version

In the original domain, to differentiate the size of a snowball, three different predicates were used. With the conversion to PDDL Numeric, these were synthesized into a single numeric function called *ball_size*. It can take three different values: 1 for small balls, 2 for medium balls, and 3 for large balls. Finally, the predicate regarding the goal was also converted into a numeric function. In the case of creating one snowman, it assumes the value 1 if the snowman has been created. This conversion takes on a different meaning for domains designed for creating multiple snowmen, as it keeps track of the number of snowmen created; in fact, it was renamed to *snowman_built*.
In this regard, two significant predicates were added:

- (snowman_at ?l - loc): location l contains a snowman;
- (ball_used_in_snowman ?b - ball): ball b is used in a snowman.

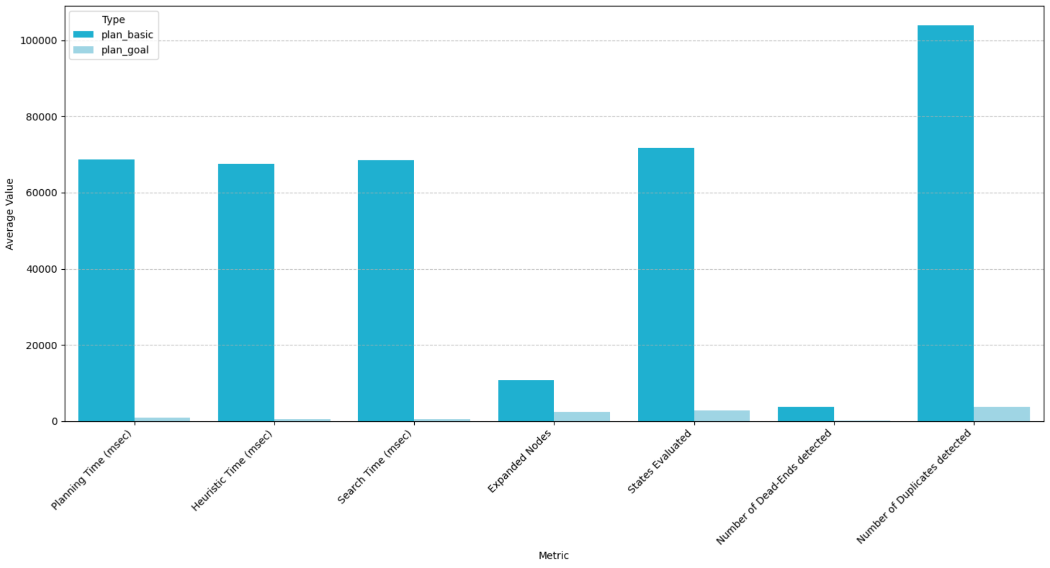
This choice was made to make the problem's objective clearer, because in the original domain, for the creation of multiple snowmen, it was checked whether all the snowballs were distinct from each other and occupied different locations in groups of three. As the domain has been implemented, it is extendable to problems designed for creating multiple snowmen.

[1]For solving the problems, the *enshp-20* solver was used, and they were solved on a device with an Apple Silicon M3 processor and 16Gb of RAM

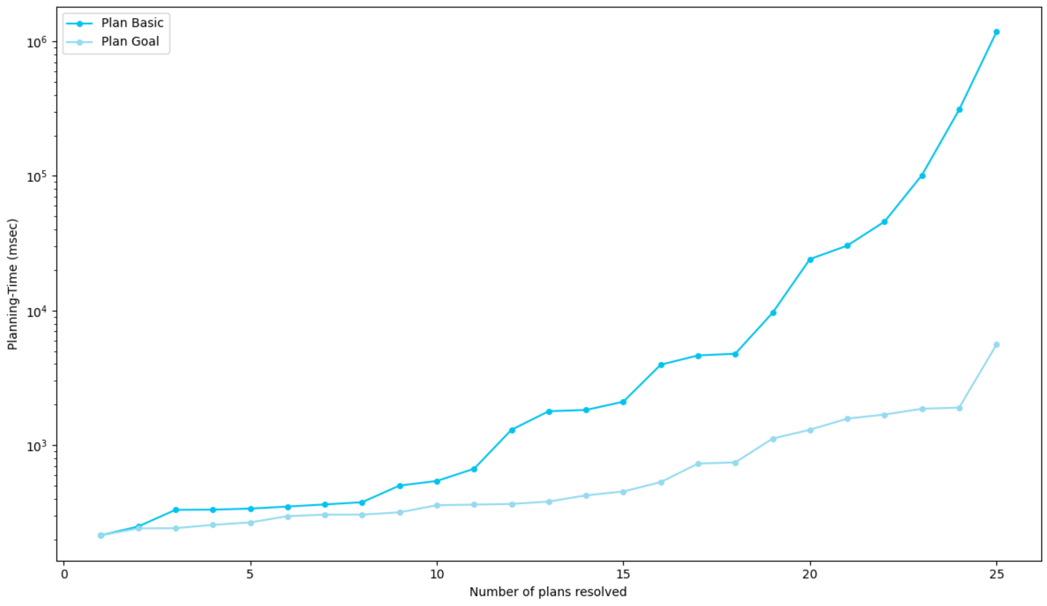
This is the simplest conversion, which reported very satisfactory results. Of the 46 proposed problems, 33 yielded a solution. In detail, we have:

Domain	**Problems**	**Solved**
1	33	31
2	13	2

Compared to the original version, there was a slight increase in grounding time, but a reduction in all other metrics.
On average, a **Grounding Time** was reported for the **Basic** version of **71.44 msec** and for the **Goal** version of **104.76 msec**, but with a reduction in **Plan-Length** of **11 steps**.



From the graph, we can see a drastic reduction in the number of evaluated **States** and **Duplicates**, which leads to a decrease in solving times, thus an optimization of the original domain.



From the following graph[2], it can be noted how the **Goal** version's **Planning Time**, as the number of solved plans increases, has a much more contained trend compared to the **Basic** version, which has an exponential trend. This is because the **Goal** version has a much smaller number of states and duplicates.

Coordinate - version

The **Coordinate** version was created with the goal of reducing the complexity of grid initialization. In the **Basic** version, the connection between the various locations is made with the use of the predicate

(next ?from ?to - loc ?d - dir)

thus requiring, for each cell in the grid, the definition of a predicate for each direction. In this version, instead, the choice was made to identify each grid cell with numeric coordinates, in order to define the locations with a numeric function.

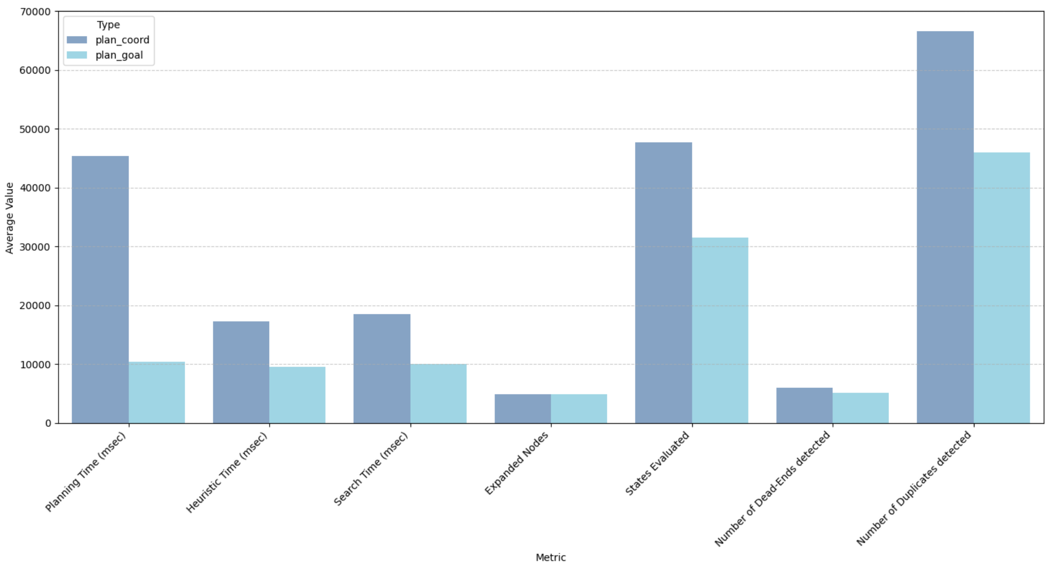
(x-coord ?l - location) and (y-coord ?l - location)

This modification is significant for reducing the number of predicates, as for problems defined on large grids, the number of predicates grows exponentially.
This version is an extension of the **Goal** version, as it maintains the previous modifications. With reference to this version, of the 46 proposed problems, 30 yielded a solution. In detail, we have:

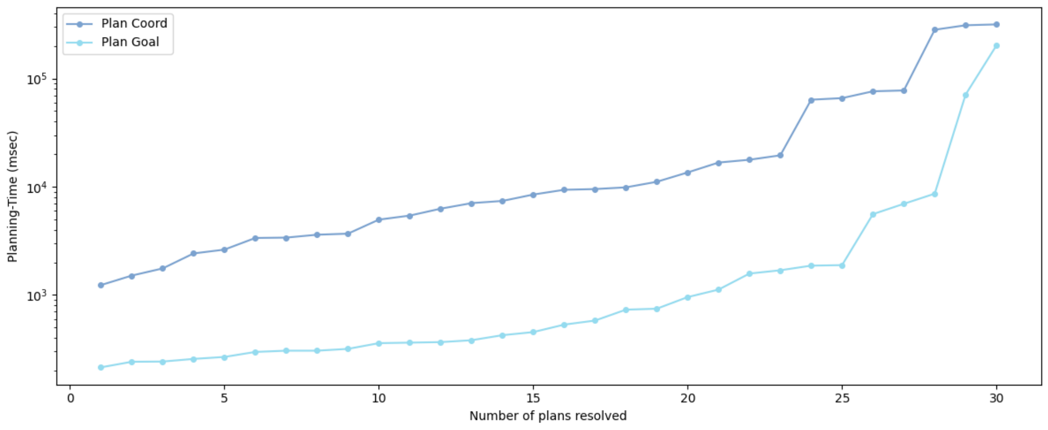
Domain	**Problems**	**Solved**
1	33	28
2	13	2

[2]This is not a comparison between the authors' plans

On average, a **Grounding Time** was reported for the **Goal** version of **113.93 msec** and for the **Coordinate** version of a significant **3170.13 msec**, a very important value to consider. Meanwhile, the plan length did not undergo significant variations.



From the graph, we can note a general increase in times. The most significant data point is the disparity in the different **Planning Times**. Specifically, on average, the **Goal** version has a better time of **4474.25 msec**, against the **46574.40 msec** of the **Coordinate** version, reporting values **10** times greater.



The following graph highlights the difference between the **Planning Time** of the two versions, emphasizing the difference in times and their trend.

Count_ball - version

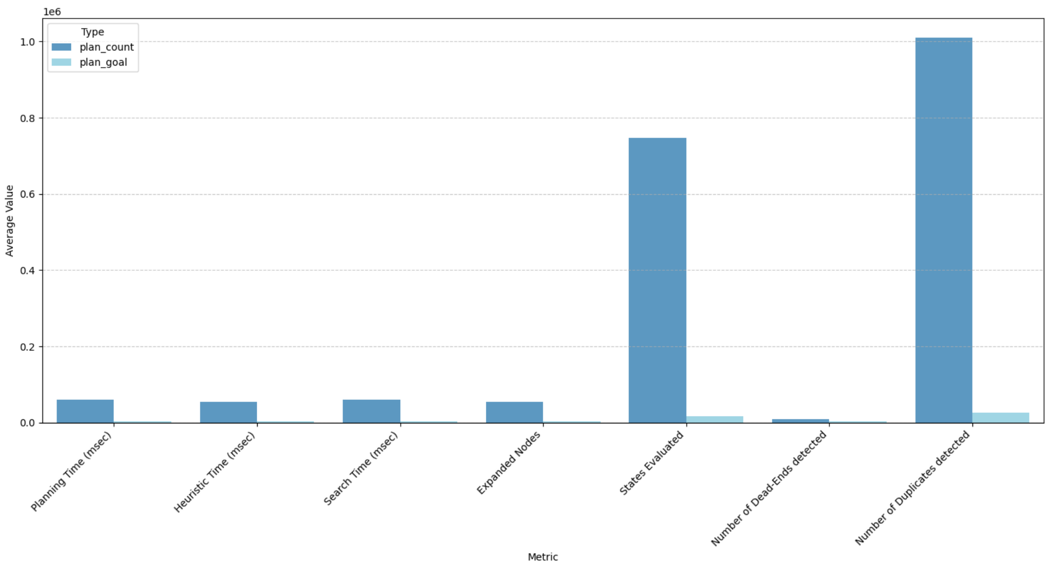
The **Count_ball** version was created with the objective of tracking the number of snowballs in each location, with the secondary aim of avoiding the goal satisfiability check. Due to the domain's structure, if three snowballs are in a single location, they will be ordered in descending size from bottom to top, ready for the creation of a snowman. With the introduction of the function

(ball_count ?l - loc)

we avoid superfluous checks, reducing solving times. This version is an extension of the **Goal** version, as it maintains the previous modifications. With reference to this version, of the 46 proposed problems, 25 yielded a solution.

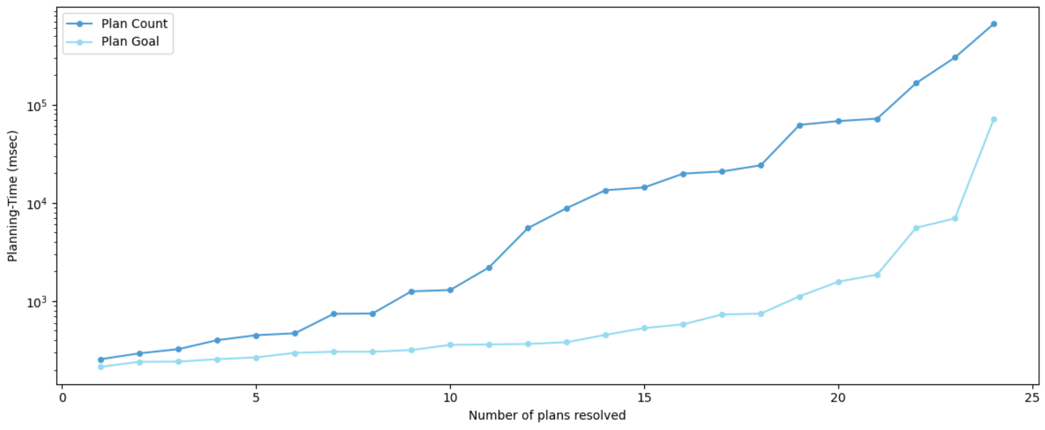
Domain	Problems	Solved
1	33	22
2	13	3

Compared to the **Goal** version, a reduction in **Grounding Time** was reported, respectively for the **Goal** version of **105.17 msec** and for the **Count_ball** version of **79.25 msec**. Furthermore, an increase in **Plan-Length** was reported; on average, the **Goal** version has a value of **52 steps** and the **Count_ball** version has **66 steps**.



From the following graph, we can highlight the disparity in **States Evaluated** and **Duplicated States Evaluated** between the two versions. In detail, we have that:

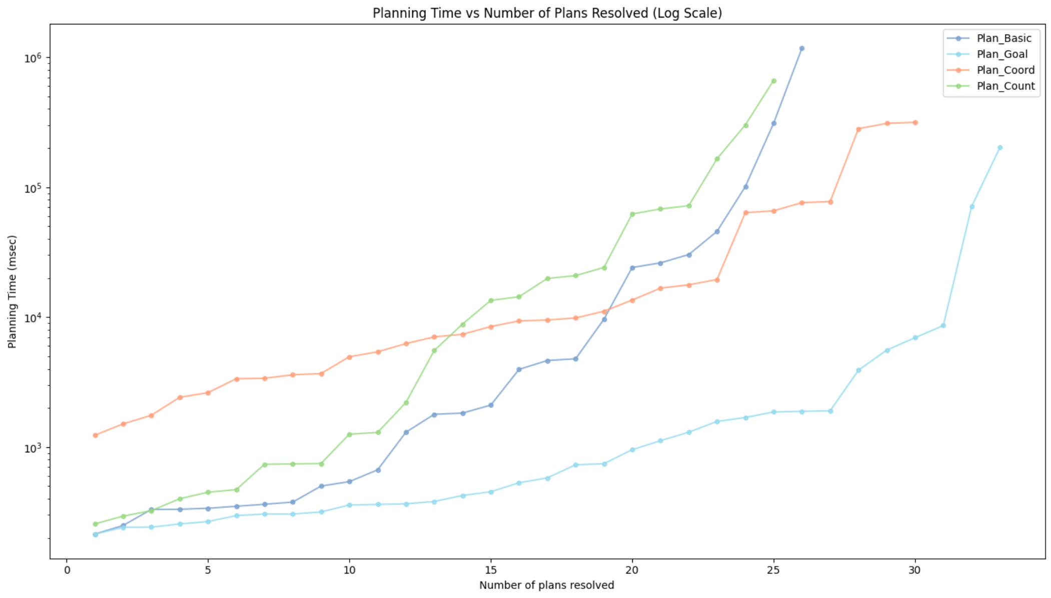
Version	States Evaluated	Duplicated States Evaluated
Goal	16,561	748,227
Count_ball	25,266	1,010,668



The following graph shows the trend of the **Planning Time** for the two versions, highlighting a more contained trend for the **Goal** version compared to the **Count_ball** version, which, on the contrary, shows higher times even at a relatively low number of levels.

General Analysis

In general, the use of PDDL Numeric has led to very satisfactory results, as an increase in the number of solved problems and a reduction in solving times were achieved. In particular, the Goal version showed an increase in the number of solved problems compared to the original version, while its extensions, namely the Coordinate and Count_ball versions, reported negative aspects that unfortunately do not make them preferable to the Goal version.
From the Basic version to the Goal version, 9 more problems were solved, and the following graph highlights the reduced times compared to the other versions.



The following table summarizes the number of problems solved for each version:

Version	**Problem 1 and 2**	**Solved**
Basic	33 - 13	25 - 1
Goal	33 - 13	31 - 2
Coordinate	33 - 13	28 - 2
Count_ball	33 - 13	22 - 3

In conclusion, each version highlights significant characteristics:

- **Basic** with particularly long plans and a relatively high number of duplicates;
- **Goal** has very fast planning with few expanded and duplicate nodes, but is not well-suited for complex scenarios;
- **Coordinate** is balanced, has a medium-long plan while managing to keep duplicates under control, with moderate planning times;
- **Count_ball** is suitable for complex plans, but has a very high number of duplicates and long times;