Benchmarking of Automated Planners: which Planner suits you best?

Wim Florijn
University of Twente
P.O. Box 217, 7500AE Enschede
The Netherlands
w.j.florijn@student.utwente.nl

ABSTRACT
Benchmarking provides an objective way to compare automated planners in artificial intelligence. Currently, there is no benchmarking framework which is able to compare the different kinds of automated planners on the market. As part of this research, a benchmarking framework has been developed which supports problems for different kinds of planners. This creates the possibility to compare both PDDL-based and Graph-based planners. In this paper a benchmarking framework which supports both PDDL-based and Graph-based planners, is described. The results of the research confirm that for the tested suite of problems, PDDL-based planners perform better than Graph-based planners.

Keywords
Artificial Intelligence, benchmark, planner, PDDL, graph

1. INTRODUCTION
Planning is a branch of artificial intelligence which concerns the generation of strategies or action sequences. Usages of planners are the realization of strategies for autonomous robots or unmanned vehicles or other smart devices. These strategies can be very complex, and thus hard to find and optimize [10].

The two most well-known kinds of planners are: Classic planners based on the PDDL language, and Graph-based planners. An example of a Graph-based planner developed within the University of Twente is GROOVE [2]. Classic PDDL based planners are often able to provide quick solutions, by not exploring the full state space [1], [9]. This is achieved by making use of efficient heuristic search algorithms. Currently, the Graph-based planner GROOVE does not offer this functionality. GROOVE however has different advantages: it is more easy and intuitive to work with, and provides additional functionality. By being able to compare GROOVE to other planners, possible areas of optimization can be found.

Currently, there are no automated benchmarking frameworks which cover both PDDL-based and Graph-based planners, let alone the whole field of automated planners. There is however a need for such a benchmarking framework. Two main groups of people can be distinguished who benefit from an automated planner benchmarking framework. These are the developers, and end-user of planners.

In the case of planner end-users, the usability of a tool is important. People will decide to use a tool if it provides the functionality they want, and if the user experience is positive. In the case of an automated planner, this user experience is largely dependent on the stability, speed and functionality of the tool. To determine and compare these factors between different planners, benchmarks or tests must be executed.

For developers, it is arguably even more important to be able to compare planners. When comparing their tool or algorithm to other planners or algorithms they can observe differences in performance. Comparing results of specific samples to the results of other planners will indicate the strong and weak points of the planner, and in which areas improvements can be made.

This need for benchmarking solutions brings us to the goal of this research: to implement an automated benchmarking framework which supports both Graph-based and PDDL-based planners. There are some factors which the framework has to satisfy, to be accepted as a valid solution.

- First of all, the framework should be extensible. This means that it should support all kinds of planners and problems, and it should be able to add more.
- Secondly, the framework should also be easy to use. It should be clear for users how the framework works, and how operations can be executed.

How these factors should be satisfied, has been determined using a literature study. Using this information, a framework has been developed which satisfies the conditions.

PAPER OUTLINE
In section 2, we will introduce the concepts of planners and benchmarks in artificial intelligence. In section 3, we will elaborate on the problem and what we want to achieve with this paper. This is followed by a description of related work in section 4. In section 5, the framework and its components are described, together with the results. Also a discussion of the results will be given in this section. Finally, a conclusion will be drawn in section 6.

2. BACKGROUND
2.1 PLANNERS
Currently, there are many different planners on the market. A lot of these planners can be categorized in two categories: Graph-based planners and PDDL-based planners. PDDL is an acronym for Planning Domain Definition Language. It works with two input files: A domain file for predicates and actions, and a problem file for objects, initial state and goal specification [5]. A Graph-based planner works with graph representations of problems and intermediate states. These

25th Twente Student Conference on IT, July 1st, 2016, Enschede, The Netherlands.

states can be visualized and edited by users. This allows for a more intuitive user experience. Examples of planning problems are described in section 5.1.1.

Different planners can make use of different search algorithms with different heuristics. The used search algorithm together with the used heuristics makes a large difference in the performance of a planner. Some planners take a short time, but can't find a short path, while other planners take more time but they try to find the shortest path.

There are various kinds of environments in which planning problems exist. In known environments, the planning can be done prior to execution (offline). However, in dynamically unknown environments, planning has to be done online, due to the fact that models and policies must be adapted when the environment changes. In this paper, offline planning will be used to validate the benchmarking framework.

2.2 BENCHMARKS

A benchmark is a standard of measurement or evaluation, and an effective and affordable way of conducting experiments [4]. It is a way to assess the relative performance of an object, by running a number of standard tests and trials against it.

Because experience has shown that information about planners is not easy to obtain, users of planning tools need guidance to select tools that are most suitable for planning problem; and researchers also desire a unified evaluation method to demonstrate the strength and weakness of their tools versus others. To make a good comparison between planners, qualitative and quantitative data should be compared.

Quantitative data can be retrieved by running the benchmark suite can be on each planner and measuring the time this takes. The shorter it takes a planner to generate a solution, the better. Then the planners can be compared based on their speed. This is however not as straightforward as it seems. There has however to be taken into account that different planners can use different (heuristic) search algorithms. Some search algorithms will find a longer path in a short time, while other search algorithms will find the slowest path in a relatively long time.

Qualitative data can be retrieved by recording the number of states which a planner has visited, before a solution is found. The amount of states is a representative solution in benchmarking, because it is hardware independent. Finally the amount of traversed paths which is needed to come to a solution is a good unit of measure. This tells us about the quality of the search heuristic. There is however a technical difficulty: not all planners will record the number of states or the amount of traversed paths it took to find a conclusion.

An important criterion for a good benchmark suite is its diversity on important characteristics of the problem. These characteristics are elaborated on in section 3.

3. PROBLEM STATEMENT

To be able to construct a benchmarking framework in which both Graph-based and PDDL-based planners can be compared, the following research questions have been defined:

1. How should a benchmarking framework be set up which is able to compare both PDDL-based and Graph-based planners?
2. Which results should the framework deliver?
3. Which format should the framework deliver the results in?

The goals of this research are to research and implement an automated benchmarking solution, so that planner tools can be tested and compared. The results of this benchmark should be made available to be used to possibly create optimizations. For the results to be valuable, a suitable test suite has to be made, which covers a range of possible scenarios. The framework together with the suite will be executed on a set of planners, to check their performance.

For a method to be scientifically correct, it must be reliable and valid. These two factors are important for the acceptance by the scientific community. Therefore the benchmarking framework should meet these conditions. The conditions are explained in more detail below.

1. Reliability means that other researchers must be able to perform exactly the same experiment, under the same conditions, and generate the same results (testability). In our case: the benchmark must return the same results when executed by multiple users under the same conditions.
2. Validity means that the results which will be obtained are valid: it is considered to be the degree to which a tool measures what it claims to measure. In our case: the results of the benchmarking framework must reflect the real performance of the tested planner.

To create a planner-covering test suite, many possible scenarios have to be kept in mind. One important factor in the performance of planners is scalability. It is very well possible that some planners are able to solve small problems relatively fast, while larger problems take relatively long.

As described in [3], other factors indicating the difficulty of a planning problem are:

1. Determinism of actions. If an action is deterministic, it will always lead to some specific state. This is however not the case with non-deterministic actions: they can lead to a choice of states.
2. Are actions probabilistic? Probabilistic planning considers that the possible outcomes of an action are not equally likely. Probabilistic planning addresses those cases where it is desirable to seek plans optimized with respect to the estimated likelihood of the effects of their actions.
3. Dynamics of the environment. The dynamics of the world may be described using discrete, continuous or hybrid models.
4. Observability of the environment. Can the current state be observed unambiguously? The actor may have to act on the basis of reasonable assumptions or beliefs regarding when the environment is not fully observable.
5. Time and concurrency. Every action consumes time. But there may or may not be a need to model it explicitly and reason about its flow for the purpose of meeting deadlines, synchronizing, or handling concurrent activities.
6. Amount of actors. Different agents may have different goals and/or metrics. In general, a multi agent planning problem can be defined as the problem of planning by and for a group of agents [11]. The behavior of agents can differ based on the problem which is to be solved. For example, in multiplayer

games, independent and self-interested agents can be used to model opponents.

To generate a benchmarking suite which delivers system covering results, preferably all these factors should be tested. However, due to the limited amount of time which is available, a couple of these factors have been selected to compile the benchmarking suite. The factors which have been selected to take into account when generating the benchmarking suite are determinism, time and concurrency and amount of actors.

4. RELATED WORK

There has been done plenty of research on benchmarks and planners. A couple of interesting papers are listed below.

4.1 PLANNERS

- Edelkamp and Rensink have explored graph transformations. The exact behavior of the Graph-based planner GROOVE is explained in [2].
- Meijer has investigated the similarities and differences of Graph-based planners and PDDL planners [6]. In this paper is explained how one can translate Graph planning problems to PDDL. Unfortunately, no implementation is available.

4.2 BENCHMARKS

- Lu et. Al. [4] have investigated the issue of benchmarking bug-detection tools. The benchmarking of bug-detection tools is comparable to the benchmarking of planners. To make a complete benchmarking suite, a benchmark has to cover a wide variety of problems. Therefore we have chosen to make use of a benchmark suite with a variety of problems, with the option to add more.

Once approximately every 2 year, a planner competition is held between planner developers. Most of these planner developers have an academic background. In the planning competitions, PDDL-based planners are tested against a suite of problems. Much information about PDDL-based planners and planning problems can be found on the website of the competition.

5. DESCRIPTION

5.1 Approach

As described in section 1, the requirements of the benchmarking framework are; extensibility and ease of use. The way in which this is achieved, and the way the framework works will be described in this section followed by a description of how the requirements have been satisfied.

The framework has been built using a layered setup. This way it is easy to add functionality without having to change large parts of the framework. A class diagram of the benchmarking framework is given in appendix C. For each class, a global description about its functionality will be provided.

- **Benchmark**: This class holds a set of planners, and a benchmarking suite. It can be used to execute the suit for some planner. Furthermore, from this class results can be written to the result writer. Planners and a suite can be added to this class.
- **Suite:** This class holds a set of problem definitions. The main method in this class is the executeSuite method. When the executeSuite method is called with some planner as argument, all problem definitions will be executed, which recursively will deliver results.
- **ProblemDefinition:** A problem definition has a name, a description and a set of problems. The idea is to be able to create different problems of the same problem definition, varying in size. This way the scalability of planners can be tested in a clear fashion. When the executeProblemDefinition method is executed, all problems which are held by the problem definition are executed, and the set of results will be returned.
- **Problem:** This is an instance of a problem definition with some fixed size/complexity. A problem can hold multiple problem instances, each for a different planning language. When the executeProblem method is executed, the problem instance which is suitable for the given planner will be executed.
- **ProblemInstance:** A problem instance is an implementation of some problem in some language such as PDDL. It does contain a ProblemType. When a ProblemInstance and a Planner have the same ProbemType, the ProblemInstance can be executed by the Planner.
- **Planner:** The planner class consists of a name, a location, a set of arguments and a regex. The name is the name of the planner, the location is the location of the planner executable on disk, and the set of arguments are the planner-specific command line arguments. The regex is the (optional) regular expression used to filter information from the command line output of the planner. An example of a piece of information which can be filtered is the amount of explored states. The planner class can be used to execute some problem instance.
- **Result:** The results class holds the results of the execution of the planner. Currently, the timestamp, execution time, output, run number, and information about the executed problem definition and planner are saved. It is however easy to extend the class with extra functionality which suits the needs of the user.
- **ResultWriter:** The result writer class writes the content of the Result class to an XLS file. This way data can be visualized and interpreted using some spreadsheet editor.
- **ProblemType**: This is a class used to match planners to problem instances. When planners and problem instances have an equal problem type, the planner is able to execute the problem instance. Examples of problem types are Graph-based or PDDL-based.

A more in-depth description of requirements and functionality of the framework is given below.

5.1.1 Extensibility

The first requirement described in section 1 tells us that the framework should support all kinds of planners and problems. As described, the back-end of the framework has been designed to support these planners and problems. Another obstacle however is, to let users input these planners and problems in a logical and user-friendly way.

To achieve this, the benchmarking framework takes an XML file as argument, where all planners, problems and other configurations can be defined. The structure of this XML file is shown in appendix B. The structure will be described below.

- **Runs**: First of all the number of times the benchmark should execute the suite is defined. It could be possible that there is a difference in execution time over different runs: this can be corrected by taking the average execution time of multiple runs.
- **Planner**: In this section, planners can be defined. A planner has a caption, location, problem type, regex and a set of arguments. The caption should be the planner name, and the location the location of the planner executable on disk. The problem type is the type of the problem it can execute. Examples are PDDL-based or Graph-based. The regex can filter the output of the planner to only get usable results. Finally the arguments are arguments to generate the command. The command which is executed on the command line can be seen as a mapping of indices to strings. Some of the indices are planner-specific arguments, and other can be problem-specific arguments. There can also be default arguments which won't change when executing different planners/problems.

  The framework uses the planners defined in the XML, to generate Planner class instances as defined in section 5.1.
- **Problem definition**: A problem definition is a description of a problem. An example of a problem is the blocks world problem. Also multiple problem instances can be created for the same problem. Each problem instance models a problem in some language.

  The framework uses the problem definitions defined in the XML, to generate ProblemDefinition class instances as defined in section 5.1.
- **Results**: In this section of the XML document, the location where the results should be saved can be indicated. One directory and one filename should be provided. The results are saved as an excel sheet.

  The results section in the XML will be used to generate a ResultWriter instance as defined in section 5.1.

To make a comparison between Graph-based planners and PDDL-based planners, a set of planners has to be selected which will be compared. PDDL-based planners have been selected based on their performance in the international planner competition, and the Graph-based planner GROOVE has been selected because it is developed at the University of Twente. The selected candidates are:

1. Graph-based planners
   - GROOVE
2. PDDL-based planners
   - MAPlan
   - PSM
   - MADLA Planner
   - MAPR and CMAP

Because GROOVE is currently the only Graph-based planner, only one Graph-based planner can be examined. All these planners have been researched and tested for usability. From the planners listed above, some would deliver incomplete results, wrong results or wouldn't execute at all. Because of the limited time available, the choice has been made to not include these planners with the framework. The planners which have been tested with the benchmarking framework are the following:

1. Graph-based planners
   - GROOVE
2. PDDL-based planners
   - MADLA Planner
   - MAPan

To make a comparison between two objects, a qualitative or quantitative method should be used. Qualitative and quantitative data gathering methods for the case of benchmarking frameworks may be hard to implement, because not all planners do deliver clear results.

In our case, to generate a covering benchmark, a combination of both qualitative and quantitative data will be used.

A benchmarking suite has been created with the following problems:

- **Blocksworld:** The rules of this problem are the following: Blocks are picked up and put down by an arm. Blocks can be picked up only if they are clear, i.e., without any block on top. The arm can pick up a block only if the arm is empty, i.e., if it is not holding another block, i.e., the arm can be pick up only one block at a time. The arm can put down blocks on blocks or on the table.

  This problem can be solved using an automated planner. In appendix A the PDDL-code and Graphs for the blocksworld problem are given. In the PDDL code, the domain file with predicates and actions (pickup, putdown, stack and unstack) is given first, followed by the problem file which defines the object, initial state and goal state. A picture of the initial state and the goal state of this problem is given in figure 1.
- **8 Puzzle**: The 8-puzzle is a square board with 9 positions, filled by 8 numbered tiles and one gap. The rules of this problem are the following: At any point, a tile adjacent to the gap can be moved into the gap, creating a new gap position. In other words the gap can be swapped with an adjacent (horizontally or vertically) tile. The objective in the game is to begin with an arbitrary configuration of tiles, and move them so as to get the numbered tiles arranged in ascending order either running around the perimeter of the board or ordered from left to right, with 1 in the top left-hand position. A picture of the initial state and the goal state of this problem is given in figure 2.
- **Logistics**: In this domain there are several cities, each one containing several locations, some of which are airports. There are also trucks, which can move within a single city, and airplanes, which can fly between airports. The goal is to get some packages from their initial locations to their destinations. Criteria can be introduced: cost and duration, and application costs and durations have been assigned to all domain actions schemas. For both criteria the lower values are preferable [7].

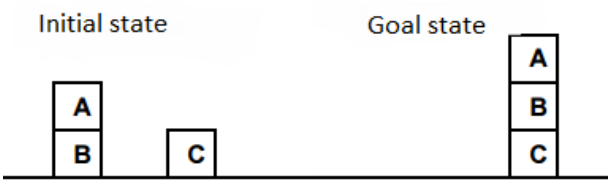

Figure 1: The initial and goal state of the Blocks world problem

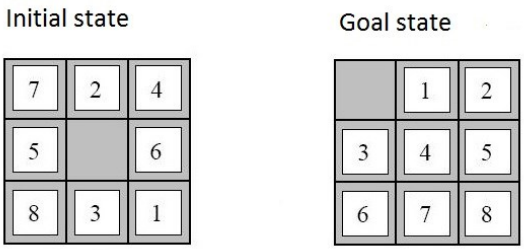

Figure 2: The initial and goal state of the 8Puzzle problem

When this benchmarking suite is executed for some planner, the execution time is measured. It is however less easy to measure the qualitative data (the amount of explored states). Some planners do keep track of this number, and return it, while other planners don't. To give users the option to record this amount when available, the planner output can be filtered and saved.

5.1.2 Ease of use

As defined in [8], usability of a system or equipment is the capability in human functional terms to be used easily and effectively by the specified range of users, given specified training and user support, to fulfill the specified range of tasks, within the specified range of environmental scenarios.

To achieve ease of use in our case, all usable components should be documented, and when possible give feedback. Two main components can be distinguished:

- The XML file containing planners, problems and other info, and the command line arguments. The XML file has been documented in this paper, and is also formally defined by the XSD file which is included with the framework.
- The command line arguments needed when executing the framework. A couple of arguments are needed when executing the framework. These are the location of the XML file, and a planner for which the benchmark should be executed. The command line interface is designed, so that it will give feedback when a user inputs faulty information.

5.2 VALIDATION

The most important result of this research is the benchmarking framework itself which has been described in section 5. However, to be able to say something about the performance of the tested planners and the quality of the framework, the framework has been used to measure the performance of the planners over the specified benchmarking suite.

The results of the different test runs are visualized and described below. Not all problems could be executed for all planners. Some planners would take too long to generate a solution; therefore a time-out limit of one hour has been used. When a planner takes longer than one hour to generate a solution for a single problem, it will be terminated and there will be no result for the problem. Another error which occurred in practice is that planners would crash while executing. In this case also no result is found for the problem. In the pictures containing bar-charts, a missing result is indicated by a missing bar.

To test the scalability of the planners, three instances of the blocks world problem have been generated. The easy variant consists of 4 blocks, the medium one of 6, and the hard variant of 12 blocks. Because the complexity of the problems increases rather exponentially, only planners with smart searching heuristics will be able to execute the hard variant of this problem within a reasonable time period.

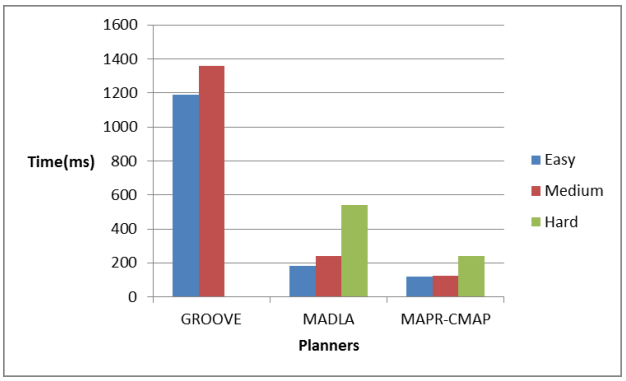

Figure 3: Execution times for the blocks world problem

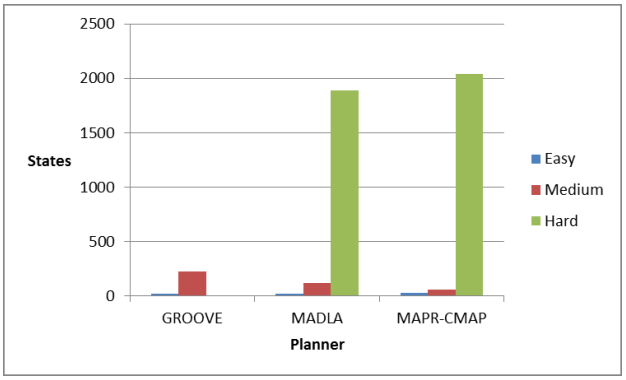

Figure 4: Explored states for the blocks world problem

As shown in figure 3 and 4; The Graph-based GROOVE planner does not perform as well as the PDDL-based planners. The PDDL-based planners take a shorter period of time, and explore a smaller number of states to come to a solution. Furthermore, the GROOVE planner could not calculate the solution for the hard version of the blocks world problem, which consists of 12 blocks. The reason for this is that the process ran out of memory. To create a level playing field, there has been chosen to not re-execute some planner while running on different hardware.

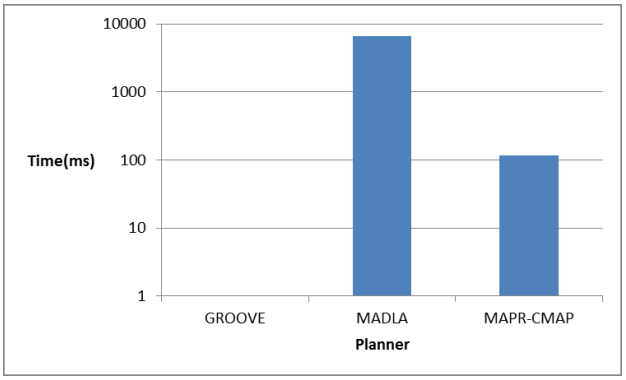

Figure 5: Execution times for the logistics problem

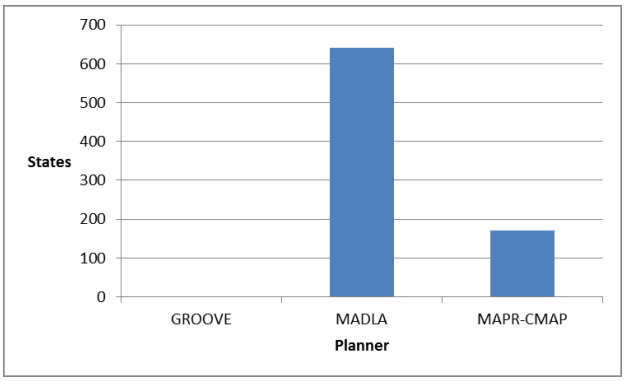

Figure 6: Explored states for the logistics problem

The PDDL-based planners were also better in calculating a solution for the logistics problem. The MAPR-CMAP planner is the fastest followed by the MADLA planner. The GROOVE planner was not able to generate a solution, because of a time-out.

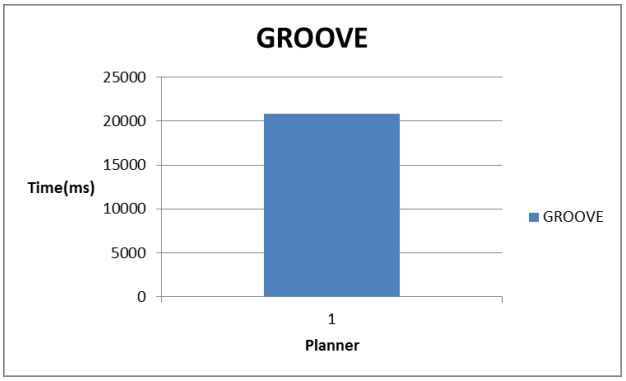

Figure 7: Execution time for the 8puzzle problem

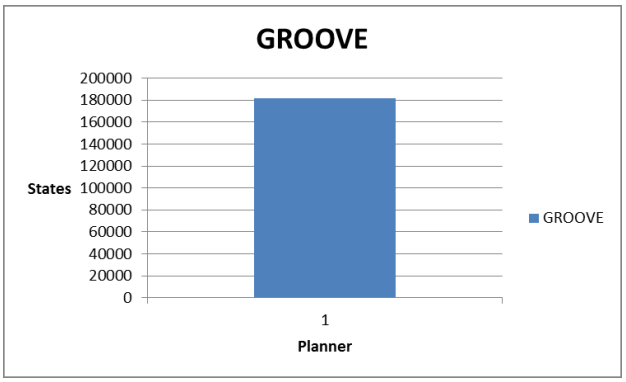

Figure 8: Explored states for the 8puzzle problem

In the case of the 8puzzle problem, the GROOVE-planner is the winner. It is the only planner able to execute the problem, because both PDDL-based planners crashed while executing.

There has also been tested how well the PDDL-based planners perform when multiple agents are used. The Graph-based planner does not support multiple agents. A custom set of blocks world problems has been created to test the performance of the multi-agent capabilities of the planners. This set contains the easy variant of the blocks world problem, with 1, 2, 3 or 4 agents. The results are that MAPR-CMAP performs better when using more agents, while the performance of the MADLA planner worsens.

The obtained results are reliable and valid, because they cover a range of problems, and vary in size. Also, when other researchers run the benchmarking suite, they will obtain roughly the same results based on the used hardware and heuristics.

5.3 DISCUSSION

The fact that the performance of Graph-based planners is not as good as the performance of PDDL-based planners can be explained by an important factor.

The tested Graph-based planner does not use as smart search algorithms as the tested PDDL-based planners do. No advanced search heuristics are used, although this will likely change in the future. The PDDL-based planners are optimized in this aspect, to deliver optimal results in the planner competition. This explains the differences in execution time and state exploration between the researched planners.

6. CONCLUSION

To conclude, it is now possible to cross-benchmark PDDL- and Graph-based planners.

A framework has been set up by looking into the similarities and differences of the types of planners, and using a layered setup. Every part of the framework has been made extensible, so it is even possible to add planners which are based on different languages.

The framework should deliver as much relevant information as possible about the executed problems. Therefore we have chosen to not only deliver the execution time, but also record information about the executed problems, timestamps, and run numbers, and the output of the planner. This way all the information is available, and the end user can choose which information to process.

By running the benchmark suite, there has been found that the performance of the tested Graph-based planner is not as good as the performance of the PDDL-based planners. Especially when the problems become more complex, the execution time of the GROOVE planner becomes very high.

We have chosen to deliver the results in XLS format. This makes it easy for the users to visualize and process the information using an (advanced) spreadsheet editor.

6.1 FUTURE WORK

Although the framework is very comprehensive, there are some extensions which could be made in future work. These are the following:

- It would be nice to have a Graphical User Interface to quickly add planners and/or Problems. This could not be done in this project, because of the limited time available. However, because of the modular setup of the framework, it is easy to add such a GUI without

having to change the whole tool. Because of the fact that the framework takes an XML file as input, such an XML file could be generated by the GUI application, and then passed to the framework.

- The benchmarking suite is currently not very extensive; only 3 problem definitions are defined. Also not all planners were able to run all problem instances. It would be nice to have a more extensive benchmarking suite with problems which cover the whole range of difficulties described in section 3.

6.2 AVAILABILITY

The source code of the benchmarking framework, including the benchmarking suite, is made available. Documentation about how to add planners or problems to the framework is given in section 5.1.1 of this paper. The framework can be downloaded from the following location:

https://github.com/WimFlorijn/PlannerBenchmark

7. REFERENCES

[1] Borrajo, D and Fernandez, S. 2015. MAPR and CMAP

[2] Edelkamp, S and Rensink, A. 2007. Graph Transformation and AI Planning

[3] Ghallab, M, Dana, N and Traverso, P. 2016. Automated Planning and Acting

[4] Shan Lu, Zhenmin Li, Feng Qin, Lin Tan, Pin Zhou and Yuanyuan Zhou. 2005. BugBench: Benchmarks for Evaluating Bug Detection Tools

[5] Mcdermott, D, Ghallab, M, Howe, A, Knoblock, C, Ram, A, Veloso, M, Weld, D and Wilkins, D. 1998. PDDL -The Planning Domain Definition Language

[6] Meijer, R. 2012. PDDL Planning Problems and GROOVE Graph Transformations: Combining two Worlds with a Translator

[7] Refanidis, I, Bassiliades, N and Vlahavas, I. 2015. AI planning for transporting logistics

[8] Shackel, B and Richardson, S. J. 1991. Human Factors for informatics usability

[9] Stolba, M and Komenda, A. 2008. MADLA: Planning with Distributed and Local Search

[10] Vlahavas, L and Refanidis, L. 2009. Planning and scheduling. [accessed 07-05-2016]. http://www.eetn.gr/index.php/about-eetn/eetn-publications/ai-research-in-greece/planning-and-scheduling

[11] Weerdt, M and Clement, B. 2009. Introduction to Planning in Multiagent Systems

APPENDIX

A. BLOCKS WORLD

A.1 PDDL CODE OF BLOCKS WORLD

Domain file:

```
(define (domain blocksworld)
  (:requirements :strips)
(:predicates (clear ?x)
          (on-table ?x)
          (arm-empty)
          (holding ?x)
          (on ?x ?y))

(:action pickup
  :parameters (?ob)
  :precondition (and (clear ?ob) (on-table ?ob) (arm-empty))
  :effect (and (holding ?ob) (not (clear ?ob)) (not (on-table ?ob))
            (not (arm-empty))))

(:action putdown
  :parameters  (?ob)
  :precondition (holding ?ob)
  :effect (and (clear ?ob) (arm-empty) (on-table ?ob)
            (not (holding ?ob))))

(:action stack
  :parameters  (?ob ?underob)
  :precondition (and (clear ?underob) (holding ?ob))
  :effect (and (arm-empty) (clear ?ob) (on ?ob ?underob)
            (not (clear ?underob)) (not (holding ?ob))))

(:action unstack
  :parameters  (?ob ?underob)
  :precondition (and (on ?ob ?underob) (clear ?ob) (arm-empty))
  :effect (and (holding ?ob) (clear ?underob)
            (not (on ?ob ?underob)) (not (clear ?ob)) (not (arm-
empty)))))
```

Problem File:

```
(define (problem blocksworld-prob1)
  (:domain blocksworld)
  (:objects a b)
  (:init (on-table a) (on-table b) (clear a) (clear b))
  (:goal (and (on a b))))
```

A.2 GRAPHS OF BLOCKS WORLD

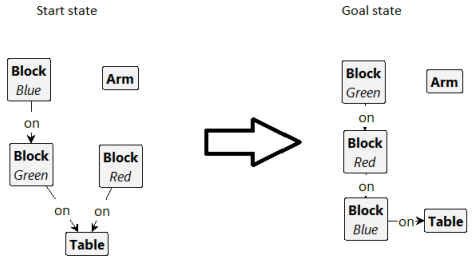

Figure 9: Goal and start graphs

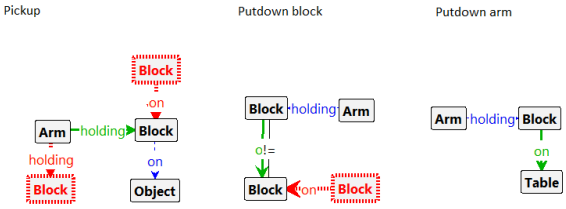

Figure 10: Rule graphs

B. STRUCTURE OF THE INPUT XML

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified" attributeFormDefault="unqualified">
    <xs:element name="benchmark">
        <xs:complexType>
            <xs:sequence>
                <xs:element name="runs" type="xs:int"></xs:element>
                <xs:element name="planner" maxOccurs="unbounded">
                    <xs:complexType>
                        <xs:sequence>
                            <xs:element name="caption" type="xs:string"></xs:element>
                            <xs:element name="location" type="xs:string"></xs:element>
                            <xs:element name="problemtype" type="xs:string"></xs:element>
                            <xs:element name="argument">
                                <xs:complexType>
                                    <xs:sequence>
                                        <xs:element name="length" type="xs:int"></xs:element>
                                        <xs:element name="defaultargument" maxOccurs="unbounded">
                                            <xs:complexType>
                                                <xs:sequence>
                                                    <xs:element name="index" type="xs:int"></xs:element>
                                                    <xs:element name="value" type="xs:string"></xs:element>
                                                </xs:sequence>
                                            </xs:complexType>
                                        </xs:element>
                                        <xs:element name="plannerargument">
                                            <xs:complexType>
                                                <xs:sequence>
                                                    <xs:element name="index" type="xs:int"></xs:element>
                                                    <xs:element name="value" type="xs:string"></xs:element>
                                                </xs:sequence>
                                            </xs:complexType>
                                        </xs:element>
                                    </xs:sequence>
                                </xs:complexType>
                            </xs:element>
                        </xs:sequence>
                    </xs:complexType>
                </xs:element>
                <xs:element name="problemdefinition" maxOccurs="unbounded">
                    <xs:complexType>
                        <xs:sequence>
                            <xs:element name="name" type="xs:string"></xs:element>
                            <xs:element name="description" type="xs:string"></xs:element>
                            <xs:element name="problem" maxOccurs="unbounded">
                                <xs:complexType>
                                    <xs:sequence>
                                        <xs:element name="probleminstance">
                                            <xs:complexType>
                                                <xs:sequence>
                                                    <xs:element name="location" type="xs:string"></xs:element>
                                                    <xs:element name="description" type="xs:string"></xs:element>
                                                    <xs:element name="argument">
                                                        <xs:complexType>
                                                            <xs:sequence>
                                                                <xs:element name="index" type="xs:int"></xs:element>
                                                                <xs:element name="value" type="xs:string"></xs:element>
                                                            </xs:sequence>
                                                        </xs:complexType>
                                                    </xs:element>
                                                    <xs:element name="problemtype" type="xs:string"></xs:element>
                                                </xs:sequence>
                                            </xs:complexType>
                                        </xs:element>
                                    </xs:sequence>
                                </xs:complexType>
                            </xs:element>
                        </xs:sequence>
                    </xs:complexType>
                </xs:element>
                <xs:element name="results">
                    <xs:complexType>
                        <xs:sequence>
                            <xs:element name="filename" type="xs:string"></xs:element>
                            <xs:element name="location" type="xs:string"></xs:element>
                        </xs:sequence>
                    </xs:complexType>
                </xs:element>
            </xs:sequence>
        </xs:complexType>
    </xs:element>
</xs:schema>
```

C. CLASS DIAGRAM OF THE FRAMEWORK