# Getting Started with JavaGP

Felipe Meneguzzi

2010-10-31

## 1 Graphplan

Graphplan is a planning algorithm based on the construction of and search in a graph [1]. It is considered a breakthrough in terms of efficiency regarding previous approaches to planning [7, 3], and has been refined into a series of other, more powerful planners, such as IPP[1] [4] and STAN[2] [5], whose efficiency has been empirically verified in several planning algorithm competitions [6, 2].

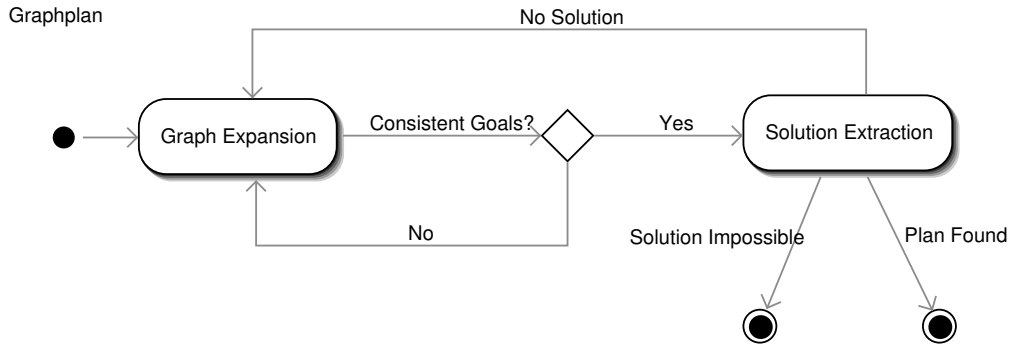


Figure 1: Overview of the Graphplan algorithm.

Planning in Graphplan is based on the concept of a *planning graph*, which is a data structure in which information regarding the planning problem is stored in a directed and levelled graph in such a way that the search for a solution can be optimised. The planning graph is not a state-space graph in which a plan is a path through the graph. Instead, a plan in the planning graph is essentially a flow in the network flow sense, which is composed of more than one path in a directed graph with the added constraint that paths must not include mutually exclusive nodes in the same graph level.

The planning graph consists of alternating levels of instantiated operators and propositions representing temporally ordered actions and the world-states that occur between the execution of these actions. Proposition levels contain nodes labelled with literals, and are connected to the actions in the subsequent action level through precondition arcs. Action nodes are labelled with operators and are connected to the nodes in the subsequent proposition nodes by effect arcs. Every proposition level denotes literals that are possibly true at a given time step, so the first proposition level represents the literals that are possibly true before plan execution (*e.g.*time step 1), the next proposition level represents the literals that are possibly true at the next time step (*e.g.*time step 2) and so forth. Action levels denote operators that can be executed at a given moment in time in such a way that the first action level represents the operators that may be executed at time step 1 and so forth. The graph contains mutual exclusion relations (*mutex*) between nodes in the same graph level, denoting that two nodes connected by a *mutex* arc cannot be simultaneously present in the same graph level for any solution. These mutual exclusion relations play a key role in the efficiency of

[1] Interference Progression Planner
[2] State Analysis

the algorithm, as the search for a solution can completely ignore any flows that include mutually exclusive nodes in a given level.

Construction of this graph is efficient, having polynomial complexity for both graph size and construction time regarding problem size [1]. This graph is then used by the planner in the search for a solution to the planning problem using data regarding the relations between operators and states to speed up the search. The basic Graphplan algorithm (*i.e.*without the optimisations proposed by other researchers) is divided into two distinct phases: graph expansion and solution extraction. The algorithm alternates execution of graph expansion and solution extraction until a solution is found or it is proven that no solution exists, as illustrated in Figure 1.

# 2 Installation and Configuration

1. Java JDK $\geq$ 5.0 is required (`http://java.sun.com`);
2. Download JavaGPfrom `http://javagp.sf.net`;
3. Decompress the binary distribution file `javagp-bin-X.Y.Z.zip`, which creates a folder `javagp-bin-X.Y.Z`
4. Open a terminal window and go to the uncompressed folder `cd /path/to/javagp-bin-X.Y.Z`
5. You can now execute JavaGPby running `java -jar javagp.jar`
6. In some cases, it might be necessary to configure the `PATH` of your system to locate the java executable.

# 3 Command Line Interface

JavaGP is a command line Java implementation of Graphplan, which takes as input a file describing the *domain* as well as a file describing the *problem*. The domain comprises the set of operators available for planning, while the problem describes the initial and goal states. A basic invocation of JavaGP, then contains references to the domain and the problem (in any order), *e.g.*`java -jar javagp.jar -p <problem file> -d <domain file>`. Syntax of the domain and problem files is described in Section 4. Besides these basic parameters, JavaGPcan also take the following two parameters:

- `-maxlevels <n>` — limits the number of graph levels to `n`, so that if JavaGPcannot find a solution with $^{(n-1)}/_{2}$ steps for a problem, planning fails.
- `-timeout <t>` — limits the time for the planner to find a solution to `t` milliseconds, if the planner cannot find a solution in that time, planning fails.

# 4 Domain/Problem Syntax

# References


[1] Avrim L. Blum and Merrick L. Furst. Fast planning through planning graph analysis. *Artificial Intelligence*, 90(1-2):281–300, 1997.

[2] Malik Ghallab, Joachim Hertzberg, and Paolo Traverso. *Proceedings of the Sixth International Conference on Artificial Intelligence Planning Systems*. AAAI, Toulouse, France, April 23-27 2002.

[3] Jörg Hoffmann and Bernhard Nebel. The FF planning system: Fast plan generation through heuristic search. *Journal of Artificial Intelligence Research*, 14:253–302, 2001.

[4] Jana Köhler, Bernhard Nebel, Joerg Hoffmann, and Yannis Dimopoulos. Extending planning graphs to an ADL subset. In S. Steel, editor, *Proceedings of the 4th European Conference on Planning*, volume 1348 of *Lecture Notes in Computer Science*, pages 273–285. Springer-Verlag, Germany, 1997.

[5] Derek Long and Maria Fox. Efficient implementation of the plan graph in STAN. *Journal of Artificial Intelligence Research (JAIR)*, 10:87–115, 1999.

[6] Derek Long and Maria Fox. Automatic synthesis and use of generic types in planning. In Steve Chien, Subbarao Kambhampati, and Craig A. Knoblock, editors, *Proceedings of the Fifth International Conference on Artificial Intelligence Planning Systems*, pages 196–205, Breckenridge, CO, USA, 2000. AAAI Press.

[7] Daniel S. Weld. Recent Advances in AI Planning. *AI Magazine*, 20(2):93–123, 1999.