A Blocks World Planner for Problem Solving Strategies

Project Report: CS 6364.001 Artificial Intelligence (Spring 2015)

By: Ninaad Pai
Email: ndp140030@utdallas.edu
Date of Submission: April 29th 2015

Abstract

Intelligent strategies are vital for solving problems occurring daily. Knowledge based strategies are required to solve problems efficiently and to obtain the desired result. This is achieved by finding a sequence of actions that lead from the initial state to the goal state of a problem. Such process is known as planning. Machines try to simulate this kind of behavior in Artificial Intelligence. This project gives an overview of current planning approaches and deals with core parts of problem solving and total order state based planning.
This project consists of a non-linear Java planner that implements essential parts mentioned above using the Blocks World domain and problems. Also, by combining some natural language processing, this planner answers questions about its own behavior i.e. when you ask 'how' or 'why' a certain step was taken, it provides a short answer about it using the concept of goal trees, also known as and-or trees.

---

1. Introduction

Problem solving and planning are relevant to Artificial Intelligence (AI), particularly in the fields of game playing, robotics, etc. It is utilized for situations where complex strategic and reasoned behavior is required.
Two key aspects are required: knowledge about problem and applied strategy (also referred as control strategy). Characteristic languages for AI are Prolog or LISP since these logical and functional programming languages also host knowledge representation and rule application. Another programming concept that is under use extensively nowadays- is object oriented programming. Java is a relatively new usage, which has proven to be very flexible, since it consists a lot of object oriented features and exception handling mechanisms. The aim of programming part of this work is to use an atypical AI programming language like Java to implement a simple non-linear total order planner.
In AI, planning systems are usually portrayed as efficient computer programs. Consistency is important i.e. the program should return stable outputs to all possible inputs.
Characteristic planning systems consist of states, operators, search procedures (or algorithms) and heuristics, as well as domain specific knowledge. Basically the roots of computational planning go back to the General Problem Solver (GPS) concept.

In planning, hard problems are approached by problem decomposition. If an entire problem is difficult to solve, it is divided into smaller tasks and sub-goals. Consequently, these have to be solved by a planner. Once, sub-goals are solved the solution is synthesized. To solve most nontrivial problems, it is necessary to combine some of the basic problem solving strategies. It is often useful to divide the problem into separate smaller pieces and to solve those pieces separately, to the extent that it is possible. Then the separate pieces must be combined together to form a single consistent solution to the original problem.

Problem solving can be described as the process of searching through a state space in which each point is corresponded to a situation that might come up. The search started with an initial situation and performed a sequence of allowable operations until a situation corresponding to the goal was reached. There are numerous ways available to move through the search space in an attempt to find a solution to a problem. For example, the A* algorithm provides a way of conducting a best-first search through a graph representing a problem space. Each node that is examined in the A* algorithm represents a description of a complete problem state and each operator describes a way of changing the total state description. For comparatively simple problems such as the 8-puzzle, manipulating the complete state description at one time is easy and reasonable.

However, for more complicated problem domains, it becomes important to be able to work on small pieces of a problem separately and then to combine the partial solutions at the end into a complete problem solution. Unless we can do this, the number of combinations of the states of components of a problem becomes too large to handle in the amount of time available.

1.1 Components of a Planning System

1.1.1. Choice of rule to be applied

The most widely used technique for selecting which rule to apply is first to isolate a set of differences between goal state and current state, and then to identify those rules that are relevant to reducing those differences. If several rules are found, a variety of other heuristic information can be exploited to choose among them.

1.1.2. Applying the chosen rule

In simple systems, applying rules is easy. Each rule simply specified the problem state that would result from its application. Now, each rule only specifies a small part of the complete problem state.

1.1.3. Detecting a deadlock

When a planning system searches for a solution to solve a particular problem, it must be able to notice when it is exploring a path that can never lead to a solution or at least appear unlikely to lead to one. The same reasoning mechanisms that can be used to detect a solution can often be used for detecting a deadlock.

1.1.4. Detecting a solution

A planning system has succeeded in finding a solution when it has found a sequence of operators that transforms initial state into goal state. In simple problem solving systems, a straightforward match of state descriptions easily answers this question. But if entire states are not represented explicitly but rather are described by a set of relevant properties, then this problem becomes more complex. The way it can be solved depends on the way that state descriptions are represented.

2. Previous Work

Blocks-world planning has been widely investigated by planning researchers, primarily because it appears to capture several of the relevant difficulties posed to planning systems. It has been especially useful in investigations of goal and sub-goal interactions in planning—particularly deleted-condition interactions such as creative destruction and Sussman's anomaly, in which a side effect of establishing one goal or sub-goal is to deny another goal or sub-goal.

Numerous book-authors such as Charniak & McDermott [*Introduction to Artificial Intelligence*, 1985], Nilsson [*Principles of Artificial Intelligence*, 1980], Norvig [*Paradigms of Artificial Intelligence Programming: Case Studies in Common Lisp*, 1992], Rich [*Artificial Intelligence*, 1983] have made their proposals and discussions regarding approaches on Blocks World problem.

Other versions of the blocks world have also appeared in the AI literature. For example, Terry Winograd's original version of the blocks world contained blocks of different sizes, and also contained pyramids.

3. Approach

This particular program is patterned after an early AI classic. It consists of a finite number of blocks (in this case, 8) on a flat surface. To make this program a bit easier to implement in order to achieve its purpose, I have kept the flat surface of width equal to all blocks' width combined. There is a simple agent, which is a robotic arm used to pick up and put down blocks. The arm can hold only one block at a time, provided that it is empty, and it can put down blocks on other blocks or on the table. There is also another constraint that blocks of larger width cannot be kept on blocks that have a smaller width, when compared to it (adapted from the concept of Tower of Hanoi).

The robot arm can perform certain actions such as:

- FindSpace- For this the arm should not be holding any other block, and the block, which it wants to pick up, should have nothing on top of it.
- Stack- Place one block on another. The arm must be already be holding a block and the surface of the target block should be clear.
- Grasp- Pick up a block from the table and hold. For this, the arm must be empty and top of block should be clear.
- Ungrasp- Put the block on table. The arm should be holding the block to be put down.

- ClearTop- Clear top of moving and target block so grasping and putting the block will be possible.

There are some more subroutines, which are involved in the actions mentioned above.
In order to specify both conditions in which an operation can be performed and the results of performing it, a few predicates are needed to be used:

- On(B1, B2): Block B1 is on Block B2.
- OnTable(B1): Block B1 is on table.
- Clear(B1): Top of Block B1 is clear.
- Holding(B1): Arm is holding Block B1.
- ArmEmpty: Arm is not holding any block.

For the question-answering module, I have used a simple goal tree, which is constructed as each step is conducted in a process, whenever we give the system a command. When we provide a command to the program say 'Put Block X on Block Y' it generates a new tree with the given command as root and all possible operations specified above as its children. The program traverses to the next node according to the control startegy, by recalling the state in which it currently is, and so on until it has reached the goal.

4. Data Set

As the two fields problem solving and planning share many components, it is difficult to differentiate between them and handle them separately. Since the emphasis is on planning, all the definitions and explanations in this section refer to a planning environment.

4.1. Predicates

A predicate consists of a single literal or relation and a set of maximally five variables or constants. It describes relations over variables, which become ground variables after initialization with valid object domains. The literal may be positive or negative. Initialized predicates are used for state descriptions; uninitialized predicates are used for operators. As given above, '*Block B1 is on table*' can be expressed as: Predicate *p1 = OnTable(B1)* in predicate form.

4.2. States

States are important and primitive components of state-based planning problems and of simple search. In terms of first-order logic a state is a proposition. It captures a situation, describing all the conditions and circumstances that are true and necessary for a clear identification of that situation in the given environment.
A state is a description of a situation, i.e. the conjunction of necessary positive predicates. The closed world assumption holds i.e. unmentioned relations are false. Therefore, all relevant predicates must be included in the state description.
For example, State s1 = (OnTable(B1), Clear(B1), ArmEmpty()).

This is a state with three predicates. All predicates are uninitialized, so the state is not instantiated. If represented in a graphical format, it will look like the following:

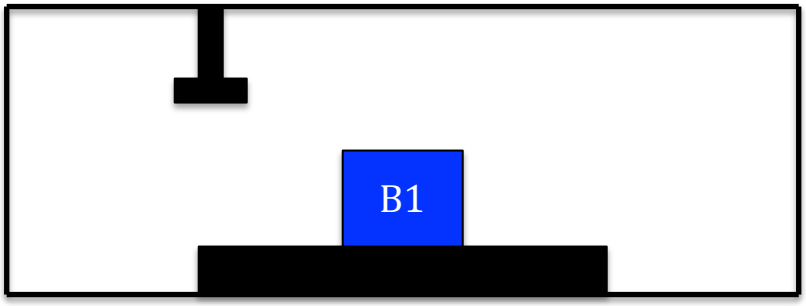

Fig 1. Representation of State s1

4.3. Operators/ Actions

Operators transform states to successor states, so that the search can be performed and a search tree can be generated. Operators use the so-called preconditions and effects. Preconditions describe the condition the state must contain or fulfill in order for the operator to be applicable. Effects describe the changes that are applied to the current state to form the resulting new state.

4.4. Operator-State Pair

Actions or instantiated operators transform a state a successor state. Therefore it is useful to associate the resulting state and the operator with its initialized parameter. This is called as operator-state pair. By using this, it is possible to trace back the states and operator that have been generated during search, since predecessor and successor states are linked to each other via a parent link.

4.5. Search Tree

The search tree, which in this case is the goal tree (also called as and-or tree) is an important data structure for problem solving, for it is used to record the whole search process accurately. Traditionally, the search tree is used to retain unexplored states only, and some other data structure or list is reserved to maintain the set of explored states. The state space of a search or planning problem contains the set of all existing states and operators that are valid within this problem. It consists of nodes (state of the environment), which are linked by edges (operator). A state and its immediately preceding operator are as an operator-state pair $p=(o, s)$ in the state space.

5 Actual Implementation

Following content gives the estimate idea of the actual implementation as shown in the demonstration. The blocks world domain can be shown using graphical demonstration, in order to avoid confusion in current state, the goal state and the state that comes up whenever a goal is reached. Given are the initial and final states of the blocks world after giving some operation to be performed.

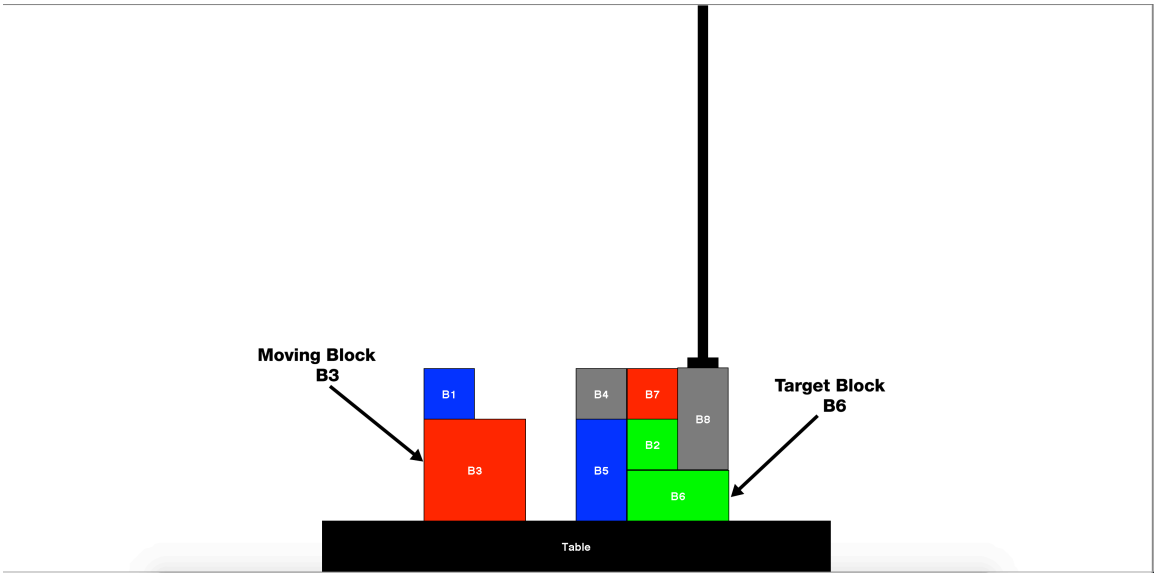

Fig 2. Initial State: Where blocks are arranged in pile

After instructing the program to move block B3 on B6, it performs heuristic search for placing other blocks on the table, filling the empty spaces from Left to Right and on the table in order to clear the top of moving block to target block. After clearing the tops of both blocks, the arm moves B3 on top of B6.

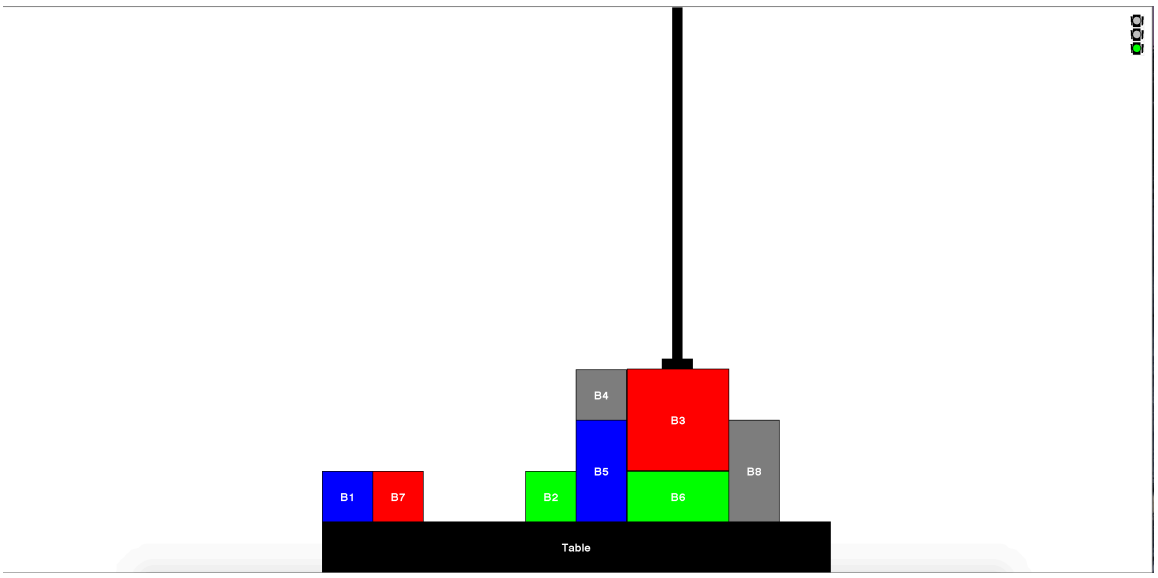

Fig 3. Final state: After moving block B3 on B6

Implementation of a search tree of a random operation say, '*put B1 on B2*' is illustrated in Figure 4 below. Consider the predicates in current state such as: Block B3 is on B1 and top of B2 is clear. The goal is given as the root of the tree and the predetermined operations as its children. The program determines which operation to be used by observing the state of the blocks world domain. After implementing the operation, it reruns the same procedure to conduct movement of blocks until the goal state is reached.

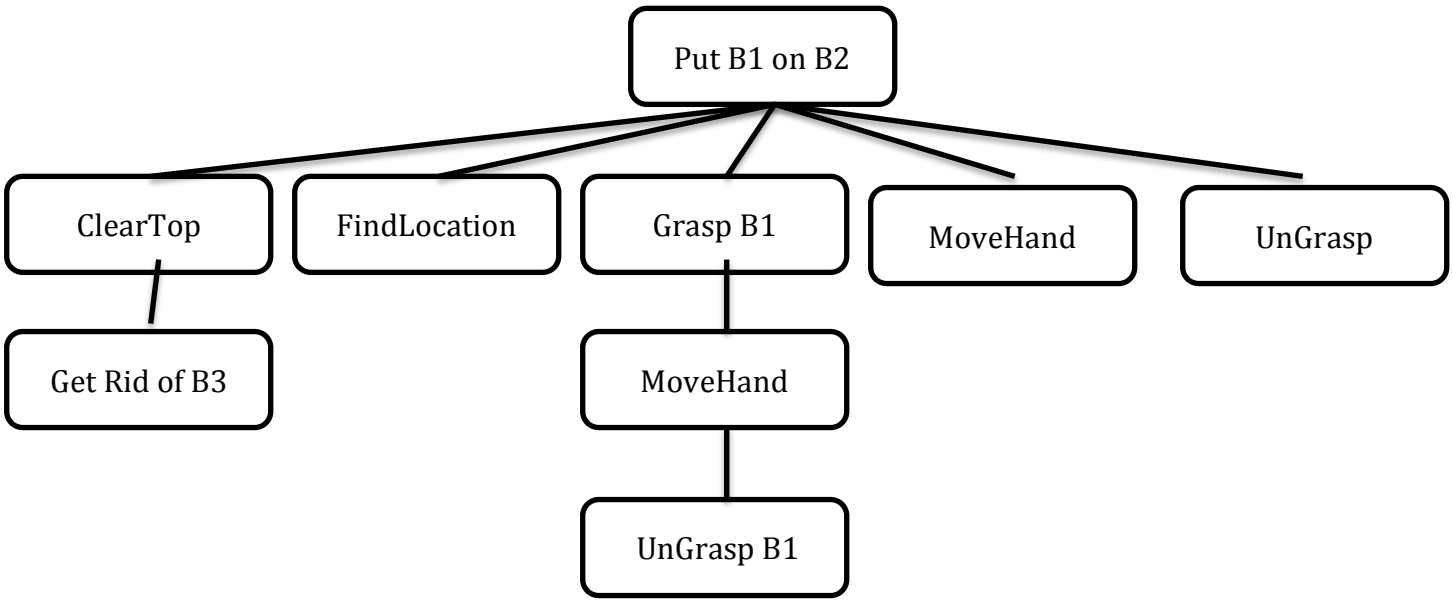

Fig 4. Search tree or Goal tree of operation Put B1 on B2

Using the goal tree, the questions asked to the system are answered, by simple traversal of the tree from bottom to top or vice-versa. Whenever the program is asked a 'why' question, for instance, '*why did you put B1 on the table?'* it checks if there is any movement recorded in the tree which involves putting B1 on the table. If it exists, the parent node of the movement is given as an answer to the question, as it simply serves the purpose of said movement.
On the contrary, if a '*how*' question is asked, for example, '*how did you keep B1 on the table?*' it traverses down the tree of the recorded movement and provides the answer. If no movement is found in the tree it returns a prompt that asks, '*did I do that?*'; which suggests that the system has not found any recorded movement in the tree.
The illustration of 'why' and 'how' question is given in Figure 5 below. The figure contains a goal tree of a state where the program is given a command to *put B2 on B3*, with precondition that, *B7 is on B3*.

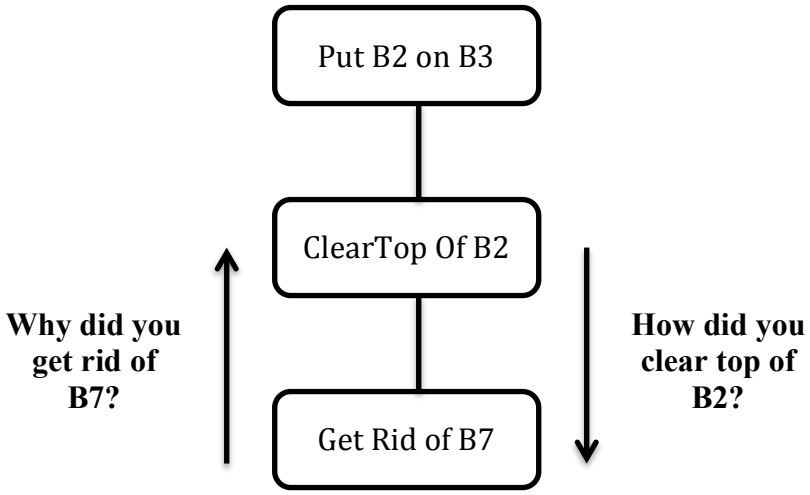

Fig 5. Illustration of question answering module by traversing goal tree

6. Conclusion

The aim of this project is to describe various aspects of planning and problem solving within the scope of AI, supported by an implementation in Java. Concerning the programming paradigm, I had various impressions and experiences. The predicate logic and planning was theoretically clear to me, but I had to refine and alter my programming approach several times. This was due to choosing Java as a programming language for a planning project. Implementing the matching of operators with states, keeping in mind which initializations were already found and which not yet- this was the most demanding part of the project that kept me busy for an entire month in the development timeline.
After completing the programming part successfully I conclude that a planning project can be implemented using object-oriented languages, but it is quite complicated and enduring. Many for-clauses, including nested ones are required to simulate matching procedures. Naturally, this slows down the program but luckily not noticeable enough for this problem.

The question-answering part was fairly new but relatively easy to understand and implement. Each question answered was basically recalled by saving the main goal given by the user and sequentially stored sub-goals or steps taken by the program to achieve the goal. Only difficulty in this module was to draw each level of goal tree, due to variable branching factors. It seems like this program can understand what you ask and answer any question you throw at it, but the Component Listener in Java is only designed to understand words such as *why*, *how*, *grasp*, *ungrasp*, *put*, etc. and block names such *b1*, *b2*, and so on. If a differently phrased question is asked it basically prompts an 'I did not understand that' remark.

7. Future Work

Possible future improvements of the program are required in the graphical representation. I have attempted this interface with the common Java Swing GUI components. Advanced frameworks such as JavaFX would definitely provide a smoother and faster approach to the visuals of this program.

8. References

1. AIPS-98 Planning Competition Committee (1998), "PDDL – The Planning Domain Definition Language Version 1.2.", 2003.

2. Fox, M., Long, D., "PDDL2.1: An Extension to PDDL for Expressing Temporal Planning Domains, 2003.

3. Knoblock, C., A., "Generating abstraction hierarchies: An Automated Approach to Reducing Search in Planning". Boston [u.a.]: Kluwer, 1993.

4. Marshall, D., "Artificial Intelligence II" (Script). Cardiff University, UK., 2003.

5. Minton, S., "Machine learning methods for planning". San Mateo, California: Kaufmann, 1993.

6. Paine, J., "AI and Problem Solving: Planning", 2004.

7. Rich, E. "Artificial Intelligence." New York [et.al.]: McGraw-Hill, 1983.

8. Rieksts, O. (2002). "Improving the State Space Search for Blocks World". Kutztown University, USA, 2003

9. Russell, S. J., Norvig P., "Artificial intelligence: A Modern Approach". Englewood Cliffs, NJ: Prentice Hall., 2003.

10. Schmid, U., "Inductive Synthesis of Functional Programs -- Learning Domain-Specific Control Rules and Abstract Schemes". Springer, 2003.

11. Schmidt, C. F., "Cognition and Computation" (Script). Rutgers University, USA, 2004.

12. Simmons, R., "Planning, Execution & Learning: Linear and Non-linear Planning" (Script). Carnegie Mellon University, USA, 2004.

13. Stillings, N. A. et al., “Cognitive Science: an introduction”. Cambridge: The MIT Press, 1998.

14. Sun Microsystems, Inc., ”API specification for the Java 2 Platform”. Standard Edition, 2003.

15. Sun Microsystems, Inc. (2003). “The Java Tutorial”, 2003.

16. Vanhornweder, K., “Blocks World Heuristics“. University of Minnesota Duluth, USA, 2003.

17. Watson, D., “Languages and Compilers” (Script). Sussex University, UK, 2001.

18.Weiss, G., “Multiagent systems: a modern approach to distributed artificial intelligence”. Cambridge: MIT Press, 2000.