**CS 5600 - Artificial Intelligence**

Report

Project 3: Planning and PDDL

Christopher H. Weber  
 11/18/2009

**Table of Contents  
  
1. PDDL Domain and Problem File(s) Description**

**2. Planner Descriptions  
 2.A BlackBox  
 2.B FastForward  
 2.C SGPlan522  
  
3. Figures  
 3.A Initial Facts per Grid Size  
 3.B Total Steps/Actions per Grid Size - Solvable Grids   
 3.C Execution Time Averages per Grid Size - Solvable Grids   
 3.D Execution Time Averages per Grid Size - Unsolvable Grids**

**4. Results and Planning Algorithms Interpreted  
 4.A BlackBox  
 4.B FastForward  
 4.C SGPlan522  
 4.D Final Thoughts  
  
Appendices  
 1.00 GIVEN  
 1.01 AI.3 Part1 – Deliverables  
 1.02 Final Domain pddl  
 1.03 Final Problem pddls  
 1.04 Solutions 4,8,16, +  
 1.05 Runtime Raw Results and Figures.xlsx  
 1.06 Solver Runtime Details - README.txt**

**1. PDDL Domain and Problem File(s) Description**

The planners used in this project accept the Planning Domain Definition Language (PDDL). PDDL consists of:

Objects – the things in the world. Can be (and are) typed.  
 Predicates – properties of objects (Boolean).  
 Initial State – the problem’s beginning state.  
 Goal Specification – things which must be true.  
 Action/Operators – ways to change the state of the world.  
  
PDDL is a relatively recent and very expressive extension and modification of STRIPS. The STRIPS (Stanford Research Institute Problem Solver) language was developed in 1971 specifically for automated planners. STRIPS and thus PDDL are computer language implementations of first order logic.  
  
The *domain\_ww.pddl* file is the domain specification and defines:  
  
 Object types: direction and square;  
 Predicates (of above objects): facing, leftOf, rightOf, adjacencyInDirection (adjD), at, isNotPit, isWumpus, isNotWumpus, isGoldLocation, has-Gold;  
 Actions: turn-left, turn-right, move, grab-gold, shoot – these actions have several parameters, and list (true) precondition predicates and effect predicate results when the action is both possible to do and is actually done.  
  
 Predicates form the substance of the 3 lists maintained by all planners used in this report: the precondition, add, and delete lists. New predicates are generated when actions are preforemd, and old predicates may be removed.

The *p<#>-<#>.pddl* file is the problem description and defines:  
  
 The objects of the world: direction and squares;  
 The true predicates of the world: directional relativity (leftOf/rightOf); adjacent squares  
and their directional relation (adjD); locations of ~pits; locations of ~/Wumpii; the initial direction and square location of the agent; the gold location;  
 And the goal: to have Gold.  
  
 Further, these files show the world’s agent, pits, Wumpii, and gold location in the Wumpus World simulators traditional output fashion for easy reference.

KEY for problem generation:

p<size>-1 – solvable, in center of grid;  
p<size>-2 – solvable, 3/4 distant from origin on diagonal;  
p<size>-3 – solvable, max distant from origin on diagonal;  
p<size>-4 – ~solvable, in center of grid, G is surrounded by Ps;  
p<size>-5 – ~solvable, in center of grid, G shares square with a P.

**2. Planner Descriptions**Base concepts:

Assertion: Any SAT or Planning algorithm is a way of solving a CSP. This assertion is not to be proven true or false, particularly in this report, but is simply an abstract thought supported by this quote:  
 “Constraint satisfaction is the process of finding a solution to a set of constraints that  
 impose conditions that the variables must satisfy. A solution is therefore a vector of  
 variables (i.e. action variables…) that satisfies all constraints.” – Wikipedia  
  
All planners in this project utilize three sets of variables: the precondition, add, and delete lists.  
  
NOTE: Material in this section is obtained by the papers on BlackBox, FastForward, and SGPlan written by the algorithms creators. To give credit where it is due, often there will be a transposition of the literal words from those documents into this report without footnote or quote markers. The references for these unattributed paraphrases and quotes are given at the bottom of each description.  
 **2.A BlackBox**   
 BlackBox implements the innovative Graphplan planner, which revolutionized the field of automated planning in 1995. However, it then converts the PDLL notation used in Graphplan into a Boolean satisfiability problem (CNF). Thus, BlackBox is a two stage planner.   
 The first stage is called plan refinement and is an implementation of Graphplan. Graphplan is a graph of variables and actions that can be fired as a result of these variables, ending at the goal variable where the problem is SAT for all actions leading to it. It uses mutexes to track the constraints on variables and actions. These mutexes are pairs of facts. Predicates/facts fully instantiated at a time-step are mutex if all the actions that create one fact are exclusive of all actions that create the other fact. Further, mutexes exist for the preconditions of each action.

“Graphplan has alternating layers of facts and actions.  
 The nodes are propositions, not states.  
 The solution is the minimal subgraph containing all goals, all supports, and no mutexes.”

The second stage is called plan extraction, and employs a Chaff search over the assignments to variables constrained by the structure of the graph. Chaff is a refined DPLL algorithm. Because Chaff accepts propositional logic sentences (as do most SAT solvers) that the current planning graph is converted into CNF before being fed to Chaff. Chaff reveals whether the problem is SAT for the depth of the current graph. Chaff employs the failed literal heuristic, random restart. It applies unit propagation on the chosen variable, and if it finds an inconsistency, sets the variable to false. Ultimately it finds the step-optimal sequence of operators (actions) that transforms the initial state to the goal state.  
 BlackBox finds optimal plans, and so is an algorithm of np-complexity. Notably, BlackBox does not use the backward chaining search of the original Graphplan algorithm. Since search dominates run-time, this is one of the major enhancements to the original Graphplan algorithm.   
 The flowchart of BlackBox is given as: STRIPS -> mutex computation -> plan graph -> CNF translate -> limited resolution/failed literal rule -> SAT engine -> solution.  
  
 Graphplan: <http://www.cs.cmu.edu/~avrim/graphplan.html>   
 BlackBox: <http://www.cs.rochester.edu/~kautz/papers/index.html> (find: BLACKBOX)  
 Chaff: <http://www.princeton.edu/~chaff/publication/DAC2001v56.pdf>  
 **2.B FastForward** FastForward is a variation of heuristic search. It does a forward search in state space which is guided by a heuristic function that works on the domain description. This heuristic function utilizes Graphplan’s planning graph method, but does not incorporate the delete list for all actions. FastForward is a local search method, uses enforced hill-climbing to escape local minima and plateaus, and identifies best successors as “helpful actions.” According to the paper cited below, “if hill-climbing does not work, then FastForward simply switches to an A\* algorithm.”  
 Its heuristic generates a “relaxed plan”, and is employed in estimating the distance to goal. “During the execution of a relaxed action sequence, states thus only grow, and the problem is solved as soon as each goal has been added by some action.” As soon as FastForward finds an action that gets it closer to the goal via its heuristic, it adds that action to its local search node, and then uses that local search node as its new starting state. The search for the successor to that local node containing the growing sequence of actions and predicates is termed a breadth-first search for shortest distance to goal. This quality makes FastForward a variation on greedy best-first search:

“Facing a search state S, FF evaluates all of its direct successors. If none of those has a better  
 heuristic value than S, it goes one step further, i.e., search then looks at the successor's  
 successors. If none of those two-step successors looks better than S, FF goes on to the three-step  
 successors, and so on. The process terminates when a state S' with better evaluation than S is  
 found… The path to S' is then added to the current plan, and search continues with S' as the new  
 starting state. In short, each search iteration performs complete breadth first search for a state  
 with strictly better evaluation.”  FastForward: <http://www.cs.toronto.edu/~sheila/2542/w06/readings/ffplan01.pdf>

**2.C SGPlan522** “SGPlan 5 partitions a large planning problem into subproblems.” It divides a goal containing more than one predicate into a single predicate subproblem/goal. Additionally, each subgoal may itself have landmarks (themselves subgoals), where all evidence shows a particular predicate must be true in order for the larger goal to be attainable. SGPlan “resolves those inconsistent (but locally optimal) solutions (of the individual subgoal solutions with their possible violation of global constraints) using our extended saddle-point condition.”   
 It is mentioned that in combining together the individual action sequences for each subgoal, SGPlan employs a robust recent variation of DPPL, not unlike BlackBox. This is surprising, given that any SAT solver would increase runtime to solution, while SGPlan was the fastest performer in runtime testing by a wide margin.  
 SGPlan uses strategies for “landmark analysis, path finding and optimization, search-space reduction, (and) evaluate(s) various heuristics for resolving global constraints.” Most significantly, it solves the aforementioned subproblems individually using a modified Metric-FF planner.

SGPlan5: <http://manip.crhc.uiuc.edu/programs/SGPlan/index.html>  
  
  
In conclusion, note that:  
  
 BlackBox uses the planning graph first demonstrated by the Graphplan planner;  
 FastForward too uses a relaxed version of Graphplan’s planner in calculating its heuristic.  
 SGPlan uses a modified FastForward (which uses Graphplan’s planner) in calculating its heuristic   
 (which itself is not equal to FastForward’s heuristic).

**3. Figures   
  
  
A.**

**B.**

**3. Figures (continued)  
  
  
C.**

**D.**

**4. Results and Planning Algorithms Interpreted**  
  
 In this section, a planning algorithms performance may reference algorithm features without greater explanation. This is because that feature will have been previously discussed in Section 2 – Planner Descriptions.

Figure A reveals a possibly interesting preliminary feature: for every doubling of the grid size, the number of initial facts/predicates quadruples.

Importantly, the figures for runtime performance are log scale over the y-axis.

NOTE: All planners returned "segmentation fault" for 32-grids except BlackBox for   
p32-1 on an 8GB RAM REX computer node. This problem contained ≈ 6000 initial predicates and created 428K nodes before completion.) The raw data on nodes created per problem size (for BlackBox only) and length of solution can be found in the appendix 1.05, remembering the KEY for problem generation**.   
  
4.A BlackBox**  
 BlackBox is most significant for its use of Graphplan, which at its first release produced step-optimal plans in small fractions of the time and space that previous planners were able to achieve. Indeed, the totals for BlackBox in Figure B represent the minimal legal step-length plans available for the given problems. It must be noted that despite the “shortcuts” that FastForward and SGPlan may be seen as taking in producing non-step-optimal plans, for these tests BlackBox was the only planner able to solve the 32-1.pddl problem without running out of memory.   
 BlackBox speeds up the original Graphplan by orders of magnitude in utilizing the Chaff SAT solver over the original backward-chaining path-finding method. However, it is still by far the slowest performer of the three planners tested (Figures C, D). The BlackBox paper does state that the vast majority of the runtime lengths for BlackBox to solution is due to the SAT solver; the generation of the planning graph (complete with all mutexes) and the conversion of that graph into CNF take a negligible amount of total runtime relative to the solver stage.

**4.B FastForward**

As Figures C and D show, FastForward is much faster in runtime than BlackBox. This is due largely to the fact that finding non-step-optimal relaxed plans has p-complexity rather than being np-hard (as is often attributed to Graphplan and its attending SAT solver). This non-step-optimality is shown in Figure B.   
 Though FF uses the same planning graph method as BlackBox, it does not employ the SAT solver step of BlackBox. Just as BlackBox’s creators’ state, it is the SAT solver which increases the run-time. A SAT solver is only important when all mutexes must be preserved in the planning graph so as to find a step-optimal plan. Thus, BlackBox and FastForward serve different purposes.

**4.C SGPlan522**  
 Though SGPlan utilizes FastForward in producing its solutions, its runtime performance is much improved over FastForward. To review the features which provide SGPlan the runtime advantage shown in Figures C and D, refer to Section 2C. Like FastForward, it does not produce step-optimal plans, but the plans it does produce are no longer than FastForward (Figure B). SGPlan’s innovative subproblem/subgoal concept alone seems a true advance in the field of AI planning. However, given that the Wumpus World problems on which SGPlan was working had only a single predicate goal; one is left to wonder the extent to which SGPlan utilized subgoal partitioning and resolution. Did it find additional subgoals/landmarks within the Wumpus World problems, particularly for size 16 grids? Or are SGPlan’s other noted refinements enough to produce the marked performance increase over its natural parent BlackBox?   
  
**4.D Final Thoughts**  
 Regarding the unsolvable grids, it would seem to be possible for a planner to ascertain that no solution is possible very early on in the search. Simply by doing a check on the node in which the gold is found, a planner could discover that because of the pits surrounding the gold, even sharing the same square, no sequence of moves could get to the gold. But this is never checked by a mutex-less relaxed plan. It is left to be encountered only when the local search finally adventures to the penultimate state(s) in which all its successors endeavor and fail to move to the final cell, the gold cell itself, or to its nearest unsafe neighbors.   
 Too, a SAT solver sidles up to the eventuality of neither T nor F for this literal sufficing. One criterion of a planner is how quickly no solution is found. If this is not a special case, if this case is often encountered in CSP, SAT, or Planning, then it is likely that future planners will conduct a cursory check on the goal states and its immediate predecessors for the no solution scenarios that were implemented in these simple Wumpus World problems.  
 As it was, FastForward took no additional runtime to discover an unsolvable problem versus a solvable one, where both BlackBox and SGPlan took 2-4 times as long to generate “no solution found”.