SIT215 Assessment 3 Report – Jacob Booth

Automated Planning and PDDL

Introduction

Automated planning, a fundamental concept leveraged by AI, encompasses the generation of action sequences that transition an initial state to a goal state. Developers use PDDL to model a problem environment by specifying goals, actions, and defining the problem domain's objects, states, properties, and relationships through types, predicates, and action schemas. This process is commonly formalised with PDDL, an industry-wide definition language for problem domain planning, which will be a key area of focus in this report. Problems and their corresponding domains are typically exposed to a solver, or some class of algorithms, where the resulting plans determine an agent action sequence for a problem-scenario. These problem scenarios may be well-defined or dynamically encountered, requiring a mix of sequential and conditional logic to convey accurately. As a concept of computational intelligence, automated planning leverages logical and heuristic methods for complex problem-solving, represented as knowledge and reasoning processes. The relationship between problem/domain representation and search algorithms aimed at identifying efficient, cost-effective outcomes, captures the essence of desired autonomous problem-solving that's employed by various AI systems. These plans can be directly executed by an agent or used to inform decision-making processes in various applications. This report will focus on the importance of clarity and refinement in the scope of automated problem planning, providing different examples of how similar problem configurations can be navigated slightly differently by employing different solvers in a basic PDDL application. Furthermore, the implications of domain complexity refinement, logical clarity and redundancies will be addressed to explain how these can be improved to accurately reflect the problem space, and how this process determines a solution's applicability. It will accompany observations made with problem examples in two different domains – a Minecraft-inspired, and a Wumpus world problem domain, both with grid-like environment representation. Their implementations leverage existing unrefined problems found at an assessment-referenced repository (https://github.com/tomsilver/pddlgym/tree/master/pddlgym/pddl) to demonstrate improved environment modelling.

PDDL Implementation

Implementation A (Minecraft-inspired world problem)

The initial Minecraft problem consisted of an intended goal state where the agent would possess a grass block in their inventory, craft a log into planks, and return to a predefined location. The initial development problem involved distinguishing the essential logical elements that served the intended focus. At first, it appeared to mainly emphasise resource acquisition and movement. After further comprehending the mechanics proposed by the default draft, it became clear the problem was more concerned with the relationship between the agent's actionable inventory mechanisms and the objects they acted upon. The inventory was defined alongside a recall action, allowing the agent to handle items in the inventory before invoking the equip action. This would simultaneously toggle the agent's `handsfree` predicate as a constraint on what actions the agent could perform. The unrefined draft also consisted of predicates like `isgrass` and `hypothetical`, signaling potential simplification by clarifying certain object relationships. A log could be represented with `islog` or `isplanks` predicates, omitting the use of hypothetical object representation for one state change. These predicates therefore became redundant and were removed to simplify the domain complexity. This strategy further distinguished object-inventory relationships from unrelated agent-inventory, agent-environment, and environment-object relationships. Furthermore, the action-specific predicates would be removed to further enforce the separation of action definitions, object state and properties, and their relationships. After the domain had been enhanced with accurate environment representation, the problem could be reasoned with less unintended consequences. To reinforce the applicability of the intended problem definition, the proposed expressions needed to be tested in logical scenarios, where relevant constraints and conditions were considered:

- Only the agent has access to the inventory and actions
- The inventory is always located at the agent
- Log blocks and grass blocks are both moveable but cannot change their own location
- The agent can `pick` a block from the environment to move it to the inventory
- The agent must be handsfree before invoking the `pick` action
- The agent can `equip` a block from the inventory
- The agent must `recall` a block in its inventory before invoking the `equip` action
- The agent must `equip` a log block before invoking the `craftplanks` action
- Using `equip` will set state to `!handsfree`

At its most basic, the problem was concerned with a goal-state transition that depended on object-specific `isLogs` → `isPlanks` property manipulation, object-inventory transitions, and agent-driven relocation to a final goal-state position. The most difficult part of the implementation would be clearly articulating the intended transition of block-object states in conjunction with the mechanisms the agent could use for object-inventory tasks. Movement constraints were not very strict for this problem, so the focus was initially placed on clarifying the order of `handsfree == true` → `pick` → `recall` → `equip` → `craftplanks` to satisfy the most complex goal first. While efficient in its representation, the solution could be made more applicable with the addition of an `unequip` action so the agent wouldn't need to collect all required blocks before crafting, as was consistent across the different plan outputs. This could theoretically be included as another action that the `recall`-ed object would be exposed to, but for the purposes of transitioning to the outlined goal states without further immediate concern, implementing `unequip` was not strictly necessary. Omitting an `unequip` action would mean the agent had to `pick` and manage items sequentially, potentially restricting the agent's ability to optimise the crafting process. This logic would not always be intentionally reflective of the problem domain across different scenarios, especially in more complex problem environments, or problem scenarios where the agent initially has something equipped. This report's applicability could benefit from considering these aspects in further revisions. Summarily, by structuring the problem around the use of object-inventory mechanisms like the `equip` and `craftplanks` actions, the agent's decision-making process was simplified to consider only the essential transitional properties for goal-state fulfilment in a basic action-sequence-planning scenario. Refining the focus on the essential elements mitigated the possibility that agent-enacted state changes would result in unintended side-effects on its environment.

Implementation B (Wumpus world problem)

The additional Wumpus world problem had several additional constraints and challenges faced by the agent. The problem environment, though similar in structure, contained potential hazardous encounters. The agent could only infer potential outcomes from cues and adjacencies, avoiding these while fulfilling goal-critical outcomes. The original revision (solution, not original draft) implemented detection actions that were invoked at different cues to conditionally interpret them, but this would be refined further. The agent should label visited squares and use adjacency cues to infer the locations of hazards, dynamically updating its knowledgebase and adapting its plan as it explored the environment, distinguishing safe cells from hazardous ones. Conditional effects would be defined in the solution's `move` action to make this possible, streamlining how the agent updated its knowledgebase to develop its interpretation of the scenario across state changes. Finally, the lethal dynamic between the agent and the Wumpus could be resolved so long as the agent possessed the arrow (default – only 1) and could shoot the Wumpus by inferring the correct location. This outcome should only be sought by the agent when there was no other way to fulfil the intended goals. If pits obstructed every alternative path, the agent could proceed to consider paths that involved killing the Wumpus. The solution included additional pits that could be comment-toggled to force a confrontation with the Wumpus, testing the domain's plan outcome in an opposing scenario. The following movement-based conditional effects were implemented to achieve this:

- When the agent moves to a `pit` location, it exhibits `dead`
- When the agent moves to a `Wumpus` location, it exhibits `dead`
- If there exists a `square` which is `adj` to the destination square that is a `gold` square, then the destination square will exhibit the `glitter` property
- If there exists a `square` which is `adj` to the destination square that is a `pit` square, then the destination square will exhibit the `breeze` property
- If there exists a `square` which is `adj` to the destination square that is a `wumpus` square, then the destination square will exhibit the `stench` property

Testing and scalability

Planned sequences produced by alternating solver types were observed across different problem definitions. Two planners were specifically selected for their applicability and differences in approach:

BFWS FF-parser solver

- Breadth-first width search algorithm tailored to parsing PDDL files
- Systematically explores the search space, considering all nodes at the current search depth before proceeding to nodes at the next depth
- Using a predefined Fast-Forward (FF) heuristic, the branching factor of solving for intended outcomes is reduced by prioritising states that are most promising for its search
- The FF heuristic estimates the distance to the goal, guiding the algorithm's path prioritization by focusing on states that appear closer to the goal based on heuristic evaluation.

This solver guarantees a cost-optimal solution by exhausting the options at each depth before proceeding. The method requires that the system's memory capacity can handle the BFWS for the given problem space complexity, proposing a significant drawback in problem domains of vast complexity and increased depth. In smaller environments with more manageable memory constraints, this serves as an ideal solution due to its exhaustive nature and straight-forward approach, despite its scaling implications. As a domain increases in complexity, requiring greater depths of generate/explore node searches, time and memory constraints would quickly become a considerable drawback. In contrast, scenarios were exposed to a LAMA-first satisficing planner, presenting its own capacity to solve automated planning scenarios:

LAMA-first satisficing planner

- Employing Landmarks, Action-Graphs, and Multi-Heuristic A*, LAMA-first is a satisficing planner – it doesn't guarantee a cost-optimal result, but one that still satisfies all required outcomes
- Heuristic-based search, employing multiple cost-based heuristics to guide the search more efficiently
- A* search algorithm is employed to balance exploration capacity with the potential to exploit lower-cost paths
- A Landmark approach, establishing sub-goals/intermediate states which progress the transition to goal states, structuring the search process

LAMA-first doesn't prioritise cost-optimality, rather focusing on finding a favorable solution with guidance from informative heuristics. This bodes well for solving in large, complex domains when compared against BFWS, as it doesn't require exhausting all solution possibilities for each search depth. By addressing a variety of heuristics and guidance mechanisms, this algorithm is much better suited to satisficing in a domain where it would be impractical to address every possible choice at each depth. This solver is therefore better suited to navigating time-sensitive situations in complex search spaces.

The refined problem file for each domain included a comment-toggle to configure 2 slightly different initial states, presenting slightly different, similarly structured scenarios. This involved altering the initial locations of grass and log blocks for the Minecraft scenario, and introducing additional pits that would force a confrontation between the agent and the Wumpus to achieve goal state fulfilment. As results were observed, it became more apparent how the definition could be improved, not simply to achieve the desired outcomes, but to imply an accurate interpretation of relationships between the agent and the problem space. The next stage would require consistent revision and improvement to ensure the produced sequences were logically justified and implicitly reflective of their real-world counterparts.

Implementation Results

Minecraft-inspired Problem

The default state configuration of the Minecraft problem was exposed to both BFWS and LAMA-first solvers, with the alternate scenario being exposed to just the BFWS solver, a result of time constraints limiting test extensivity. Despite this rather minimal approach to testing, the results consistently reflected the nature of each algorithm under the complexity of the domain constraints. Minecraft scenario A output resulted in plans of equal steps, though differing slightly in the selection of where to `move` to at a specific branching moment, reflecting the potential of LAMA-first's guidance algorithms to still result in cost-optimal outcomes without being bound by the prioritisation and exhaustive method used by BFWS-FF. LAMA-first was able to find a cost-optimal solution while expanding less nodes. This demonstrated the explorative search LAMA-first uses, and the exhaustive approach BFWS employs. Furthermore, the total nodes generated for both scenarios A and B by the BFWS were the same, outlining the space complexity in relation to environment navigation – the more stops made, the larger the space requirements to maintain the list of nodes that still need to be explored. The execution times showed the most divergence, where the BFWS was significantly faster than the LAMA-first planner. The results are in line with the algorithmic complexity of each. The LAMA-first planner consistently performed a greater number of computations given the maximum search depth, employing different mechanisms and heuristics to guide the search in a way that reduces memory constraints associated with breadth-first search-depth retention in complex environments. The results outlined the need to balance problem space complexity, time/space complexity, and search-space complexity when automating planning processes. For the purposes of this smaller problem space, BFWS was time-efficient while retaining a much larger number of generated, yet unexpanded, nodes. This approach would likely become impractical when applied to a problem domain that more closely resembles realistic problem space complexity. With that clarified, it's safe to assume that more comprehensively modelled scenarios of greater complexity and constraints would benefit from a well-guided approach like LAMA-first. For the minimal scale of the example explored, BFWS was undeniably the fastest and computationally optimal solution.

Raw Minecraft Problem Time/Space Metrics

	Scenario a BFWS	**Scenario a LAMA-first**	**Scenario b BFWS**
Total Time (s)	0.000260001	0.00823749	0.000266999
Plan Length (steps)	13	13	13
Plan Cost	13	13	13
Nodes Generated	110	73	130
Nodes Expanded	36	17	38
Relevant Atoms	N/A	207	N/A
Total Queue Pushes	N/A	389	N/A

Raw Minecraft Problem Search-Space Complexity Metrics

	Scenario a BFWS	**Scenario a LAMA-first**	**Scenario b BFWS**
# Actions	58	N/A	58
# Fluents	47	N/A	47
Initial Landmarks	N/A	15	N/A

Goal Landmarks	N/A	3	N/A
Disjunctive Landmarks	N/A	2	N/A
Conjunctive Landmarks	N/A	0	N/A
Total Mutex Groups	N/A	3	N/A
Mutex Groups Size	N/A	8	N/A
Registered States	N/A	18	N/A
Evaluated States	N/A	18	N/A
Dead Ends	N/A	0	N/A
Int Hash Set Load Factor	N/A	18 / 32 = 0.5625	N/A
Int Hash Set Resizes	N/A	5	N/A

Wumpus World Problem

The LAMA-first planner was the chosen double-up for the Wumpus domain. The stricter nature of movement required multiple conditions at each `move`, so the LAMA-first approach seemed ideal to test across both lethal and nonlethal scenarios. In this relatively simple problem space, LAMA-first was still likely to be beaten on execution time by the BFWS solver considering the computations required from the initial search-depth onward, in proportion to the problem complexity. It was hypothesised that by being exposed to slightly greater movement conditions, the LAMA-first planner would require less generated/explored nodes than the alternative. The results of the Wumpus world tests did not support this, where the generated/explored nodes of the nonlethal problem-scenario were markedly similar between solvers, resulting in comparable generation/exploration requirements for each solution-search. This suggested that increased branching from conditional effects was not supported by the results. Contrastingly, the inclusion of conditional effects may have distinguished agent-movement-hazard relationships in the solution knowledgebase, distinguishing agent-movement from agent-location, agent-hazard, and hazard-location relationships. The lethal scenario included additional pits in the initial state via comment-toggle, resulting in a problem-scenario for which no plausible action sequence that didn't involve killing the Wumpus could be found. The search metrics reflected this in the number of nodes generated, where the nonlethal LAMA-first solution had a 24:7 generated-expanded node ratio in comparison to the lethal scenario's 18:7 generated-expanded node ratio. These results reflect the simplification of the problem-space upon removal of the Wumpus threat. The nearest cue to the agent's goal required less exploration to encounter after-the-fact, requiring less consideration of additional subsequent states that would progress the intended goal transition while simultaneously avoiding the Wumpus in adherence to the domain constraints. The use of heuristics became more effective as the search-space complexity simplified, resulting in an optimal solution being found while accounting for slightly less outcomes after the point threat-removal.

Raw Wumpus Problem Time/Space Metrics

	Scenario a LAMA-first	**Scenario a LAMA-first**	**Scenario b BFWS**
Total Time (s)	0.00721225	0.00638101	0.000129
Plan Length (steps)	6	5	5
Plan Cost	6	5	5
Nodes Generated	18	24	21
Nodes Expanded	7	7	8
Relevant Atoms	257	255	N/A
Total Queue Pushes	1150	1020	N/A

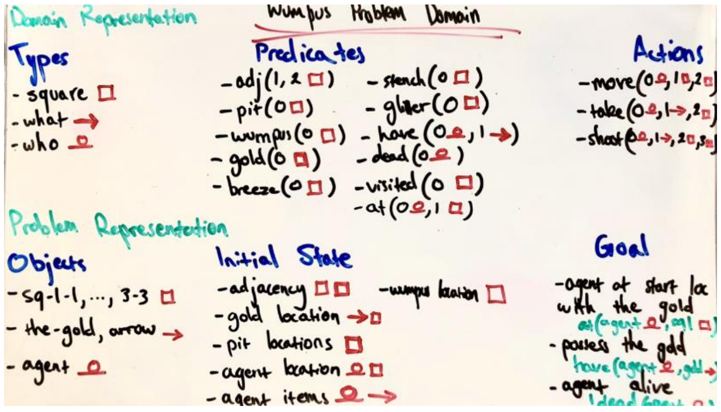
Raw Wumpus Problem Search-Space Complexity Metrics

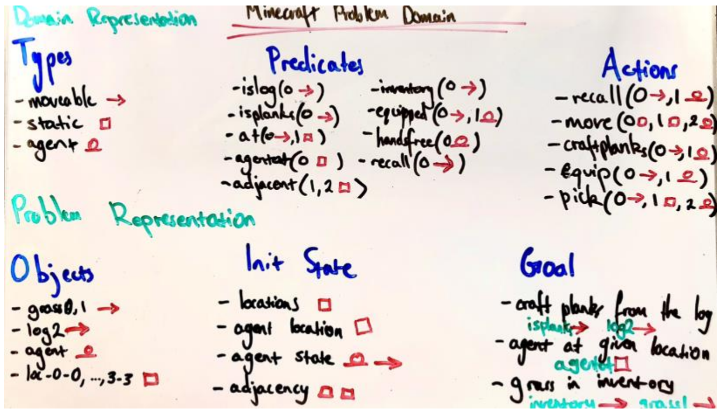
	Scenario a BFWS	**Scenario a LAMA-first**	**Scenario b BFWS**
# Actions	N/A	N/A	52
# Fluents	N/A	N/A	47

Initial Landmarks	4	2	N/A
Goal Landmarks	2	2	N/A
Disjunctive Landmarks	1	2	N/A
Conjunctive Landmarks	0	0	N/A
Total Mutex Groups	0	0	N/A
Mutex Groups Size	0	0	N/A
Registered States	8	8	N/A
Evaluated States	8	8	N/A
Dead Ends	0	0	N/A
Int Hash Set Load Factor	8 / 8 = 1	8 / 8 = 1	N/A
Int Hash Set Resizes	3	3	N/A

The contrasting complexities of each solver were inferred by the results for each scenario by their execution metrics and implied search-space complexities. This was consistent across both problem domains. While the BFWS solver consistently found the most optimal path quicker, the LAMA-first solver took much longer, substituting optimal execution time for guided solving that would work better in complex problem spaces. This suggested that for the scale of these simple problem spaces, BFWS was ideal, being best suited for solution optimality where there were no notable memory constraints. If the complexity of these domains were to grow, requiring vastly increased solution search depth, it would quickly become impractical to use BFWS. If the agent were required to make more complex decisions in time/memory-constrained scenarios, solving would very likely benefit from robust guidance mechanisms used in the LAMA-first satisficing planner, providing better structure and precision to complex searches where it would be impractical to search as exhaustively as BFWS. Additionally, the output sequences and their search metrics supported the validity and reflective accuracy intended by the problem domain definition.

Knowledgebase Representation





The knowledgebase representation between Minecraft and Wumpus world domains presented distinct problem planning challenges due to the unique focus and environmental knowledge implied by each. The Minecraft problem was largely concerned with the interpretation of inventory mechanisms (actions) and their relationships to objects that could be acquired in the problem environment. Represented by 4x4 2D grid-like spaces, each environment's navigational representations implied adjacency relationships as a constraint on traversal logic. This is effective when considering the context which program output may be observed under, emphasising that decision-making optimality is enhanced by thoroughly articulating action requisites. Additionally, the representation of `moveable` types for objects that could be shifted from the environment to the agent's inventory served as a mechanism for clarifying objects that did not need to be actioned upon at their initial locations. This enhanced logical clarity by distinguishing immoveable grid locations from their corresponding acquirable resources (if present). The Wumpus world problem was slightly different, requiring thorough clarification of movement-effect conditions based on the presence or absence of accumulated state properties. The conditional effects associated with `move` were constructed around the agent's need to address each cue upon arrival, where cues were only inferred by the movement of the agent to a goal-critical adjacency. The Minecraft problem was largely concerned with interpreting goal state requirements in conjunction with agent-inventory-object actions. The location of resources that these actions would be carried out on was assumed to be known for each scenario in this domain. The Wumpus world demanded strict avoidance of hazards and exploration of the environment to infer a required location. A `visited` predicate was used to represent safely navigated cells in a dynamically updated knowledgebase, implying the interpretation of a problem space filled with hazards and safe spots of initially unknown variation, in real time. Additionally, the use of logical preconditions, emphasis on predictable state transitions and nuanced object/state relationships that were logically reflective of the elements they were modelling, dynamic environment representation was used to better align the solution with automated planning principles.

Reflection and Conclusion

The concepts associated with automated planning present a considerable paradigm shift from other program-based problem-modelling. Implementing automated planning concepts requires the use of tools like predicate logic, Boolean logic, inferred knowledge, and logical expressions, guided by principles including dynamic environment representation and complexity simplification for problem modelling in different scenarios. These tools optimise sequential adherence for problems requiring an agent to transition initial environment states to intended goal states. The domain's permissible actions, object properties and implied relationships require careful modelling to facilitate state changes without logically inferring potentially unwanted side-effects. Dynamic environments need to be represented in ways that allow action sequences to be constructed based on what is implied and inferred by accumulated state. This approach leverages automated planning principles for agents faced with initial problem configurations containing potentially unknown goal-critical requirements. This assessment report was beneficial in clarifying automated planning concepts, which require a different frame of reference from programming action sequences in a pragmatic, functional sense. Instead of focusing on the way a sequence of actions might be carried out, automated planning involves modelling representations of actions, requirements, effects, and relationships that are bound by domain-specific logic in a problem environment. It requires developers to ask a different question to, "How might this sequence be enacted?", instead, considering the following: If there exists an optimal sequence for transitioning a problem-environment's initial state to intended goal-states, while adhering to domain-specific constraints and actions, what is implied and inferred by the order of the optimal sequence? What individual logical expressions must be defined for a proposed sequence to be deemed a valid solution? Automated planning principles refine the complexity of how said expressions are modelled and represented, with human-readable mediums to structure the logic as problem domain definition applications.