**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 can model a problem environment, specifying goals and actions, defining, and distinguishing a domain/problem’s objects, state, properties, and relationships, often formalised with PDDL, an industry-wide definition language for problem domain planning. 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 optionally be adopted in a variety of different ways, from being collected as training data for a machine-learning model, to serving as sequence plans considered by any higher-order decision-maker. 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 with the 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 across 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 THE LINKED REPO to demonstrate improved environment modelling.

**PDDL Implementations**

**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 would need to be defined in conjunction with a `recall` mechanism so the agent could consider a specific item in their inventory on which to invoke `equip`, simultaneously toggling the agent’s `handsfree` predicate as a constraint on what actions the agent could perform. The 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 and 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 can 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 in this way implied that all blocks would need to be `pick`-ed before `craftplanks` could be invoked (alternatively, bypassing `handsfree` as a prerequisite for `pick` and `craftplanks` could be explored). This logic would not always be intentionally reflective of the problem domain across different scenarios, especially where the agent already has something equipped, or in more complex problem environments. The solution could be made more applicable by 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 an 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. While the solution would benefit from additional logic, these implementations were not crucial to this report.

**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. Considering the dynamic nature of accumulating cue-location knowledge to infer the potential location of hazards, the agent should only be initially privileged to cell adjacencies, imparting the dimensions of the problem environment from a navigational perspective. The agent could only infer potential locations from cues and adjacencies, avoiding these while fulfilling other critical goal outcomes. The most immediate solution might be to enforce detection actions that are called on different squares to dynamically detect the cues and interpret them accordingly, but this could be refined further. If the agent could accumulate a knowledge of visited squares while avoiding hazards based on adjacency cues, the search process would dynamically generate solutions to accumulate knowledge of the problem configuration across state changes. Conditional effects would need to be defined in the solution’s `move` action to make this possible. This ensured cues could only be addressed as they were encountered, and their implications considered accordingly. This approach aimed to enforce dynamic environment representation, requiring navigation without initial knowledge of all relevant states. This would see the agent updating its knowledgebase to develop its interpretation of the scenario across guided 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 involve killing the Wumpus. This problem required direct consideration of bidirectional adjacency with some locations simultaneously serving as cues for hazards and goals. 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.

**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 helps in estimating goal distance, guiding the algorithm’s path prioritisation method

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 the most cost-optimal result
* 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 planning problems.

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**

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 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 refinement LAMA-first emphasises, 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. BFWS uses a straight-forward, layer-exhaustive approach backed by the fast-forward heuristic as the search complexity increases in depth. The LAMA-first planner performs a much greater number of computations, employing different mechanisms and heuristics to guide the search in a way that reduces memory constraints associated with breadth-first search-depth retention. The results outlined the need to balance problem domain complexity, time/space complexity, and algorithmic complexity when automating planning processes. For the purposes of this smaller problem space, BFWS was efficient in terms of time complexity 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.

CAPTURES OF RESULTS

The LAMA-first planner was chosen as the ‘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, clarifying plausible path logic from agent-movement and agent-hazard 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 imply 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 when the agent eliminated the threat.

CAPTURES OF RESULTS

The use of mechanisms employed by each solver were inferred by the results for each scenario through their execution metrics and 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 strict memory requirements weren’t a constraint. If the complexity of these domains were to grow, requiring increased solution search depth, applicability and execution requirements would quickly become imbalanced when automating plans with a BFWS solver. 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 as a 4x4 2D grid-like space, the space/environment representation imposed 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 resource (if present). The Wumpus world problem was slightly different, requiring thorough clarification of movement-effect conditions based on the presence or absence of immediately known state properties. The conditional effects associated with `move` were constructed around the agent’s need to address each cue uniquely. 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 agent demanded strict avoidance of hazards and exploration of the environment to infer the goal’s location. Hazard locations could only be inferred by their adjacency cues, implying stricter navigational constraints with less initial known state. 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. The emphasis on survival and exploration was captured by enforcing conditional action effects, logical preconditions, predictable state transitions and nuanced object/state relationships that were logically reflective of the elements they were modelling. This involved leveraging dynamic environment representation to better align the solution with automated planning principles given the domain constraints.

**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 expressions, guided by principles including dynamic environment representation and complexity simplification for problem modelling in different scenarios. These considerations 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 extremely beneficial in clarifying automated planning concepts, which require a paradigm shift 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, 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 help refine the complexity of how said expressions are modelled and represented, with human-readable mediums to structure the logic as problem domain definition applications.