**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 the environment, specifying goals and actions, defining and distinguish 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 parsed by a solver or class of algorithms, the resulting plans determine how problems are interpreted by the agent under specific constraints in scenarios that are either well-defined or dynamically encountered. As a concept of computational intelligence, automated planning leverages logical and heuristic methods for complex problem-solving, represented as knowledge and reasoning processes. It serves as a critical foundation for logical action sequences in environments where an AI system would be tasked with transitioning to intended goal states based on an initial problem configuration. This relationship between problem/domain representation and computationally robust calculations aimed at achieving the desired outcomes as efficiently as possible, captures the essence of desired autonomous problem-solving that’s employed by various AI systems. These plans can additionally be adopted in a variety of different ways, from being collected as training data for a machine-learning model, to serving as plausible 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 to parse the PDDL application. Furthermore, the implications of domain complexity refinement, logical clarity and redundancies will be addressed to explain how these can be improved for applicability to accurately reflect the problem space. It will accompany observations made with problem examples across two different domains – a Minecraft-inspired and a Wumpus world problem domain (both grid-like environments). 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, then return to a predefined location. The initial development problem was distinguishing the essential 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 were to act 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 can perform. The draft also consisted of predicates like `isgrass` and `hypothetical`, signaling potential simplification by enhancing and clarifying certain object relationships. A log could be represented with `islog` or `isplanks` predicates, omitting the use of hypothetical object representation for a simple state change. These predicates therefore became redundant and were removed to simplify the domain complexity. This strategy further refined and 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 addressed with less unintended consequences moving forward. To reinforce the applicability of the intended problem definition, the established rules needed to be tested in logical scenarios, where relevant constraints and conditions were considered:

* Only the agent has access to the inventory and all 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 a specific crafting task, object state transitioning, 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 placed on clarifying the order of `handsfree == true` 🡪 `:pick` 🡪 `:recall` 🡪 `:equip` 🡪 `:craftplanks` to satisfy the most complex goal first. While robust and efficient in its representation, the solution can be made more applicable with the addition of an `unequip` action so the agent doesn’t need to collect all required blocks before crafting, as was consistent across the different plan outputs. 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 still implied that all blocks would need to be `pick`-ed before `craftplanks` could be invoked (alternatively, bypassing `equip` 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 to the essential elements mitigated the possibility that agent-enacted state changes would result in unintended side-effects.

**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 path of visited squares while avoiding hazards based on 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 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 goal. If pits obstructed every alternative path, the agent may 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 also provided 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 for one scenario in each problem space:

**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 the domains increase in complexity and require multiple depths of acquire/craft sequencing, execution time would quickly become a considerable drawback. In contrast, scenarios were exposed to a LAMA-first satisficing planner, presenting its own capacity to solve in 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 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 stagger the transition to the final goal state, structuring the search process

LAMA-first doesn’t prioritise 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. 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 search depth. This solver is therefore better suited to navigating time-sensitive situations in complex planning problems.

**Implementation Results**

The strengths and weaknesses of each solver were somewhat reflected in the computational requirements for each scenario. Where the BFWS solver consistently found the most optimal path quicker, the LAMA-first solver took slightly longer, and in one scenario required increased plan cost to achieve the desired outcomes. This suggested that for the scale of complexity in these simple problem spaces, BFWS was ideal, being best suited for solution optimality where strict memory requirements are not a constraint. If the complexity of these domains were to grow, requiring increased planning depth, the balance between solution applicability and execution time would quickly become imbalanced when using the BFWS algorithm. If the agent were required to make more complex decisions in time/memory-constrained scenarios, the solution would likely benefit from the more 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 would.

**Knowledgebase Representation**

The knowledgebase representation between the Minecraft and Wumpus world scenarios presented distinct problem planning challenges due to the unique focus and environmental knowledge implied by each domain. 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 2D grid-like space, the space/environment representation imposes adjacency relationships as a constraint on traversal logic. This is effective when considering the context the plan output will be observed under, emphasising how 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 do 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 consequences based on the presence or absence of certain properties at each cell. The conditional effects associated with `move` were constructed around the agent’s need to address each queue uniquely. The Minecraft problem was largely concerned with interpreting goal state requirements in conjunction with inventory-based object-focused actions. The location of the resources 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 knowledge. A `visited` predicate was employed to determine 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. The constraints associated with avoiding hazards at unknown locations required conditional consideration based only on what was immediately clarified on a cell-by-cell basis. 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 constraints.

**Reflection and Conclusion**

The concepts associated with automated planning present a considerable paradigm shift from other program-based problem-modelling. PDDL and automated planning concepts emphasise the use of logical tools like predicate logic, Boolean logic, inferred knowledge, and principles including dynamic environment representation and complexity simplification. These considerations aid in optimising sequential logic adherence for problems requiring an agent to transition its environment’s initial states toward intended goal states. The domain’s permissible actions, object properties and implied relationships require careful modelling to facilitate state changes that don’t logically infer potentially unwanted side-effects in the way a solution is approached. Additionally, dynamic environments need to be represented in a way that allows action sequences to be constructed based on state inference. This approach leverages automated planning principles for agents faced with initial problem configurations containing potentially unknown goal-critical state.