# KBAI - ICA 1 Pitch Notes

## Introduction

Our work will look at the functionality provided by planner and search-ops. We will also be evaluating and comparing the efficiency of these searching algorithms against various test scenarios.

## Test Variables

The variables in our tests include:

* Starting state size.
* Goal state size.
* Tuple size.
* Tuple order in state.
* Tuple order in operators.
* Duplicate tuples
* Quantity of operators.
* Quantity of unique keywords in the state.
* Test machine.

## Performance & Utility Analysis

The following metrics will be used to evaluate the performance of the searches:

* Correctness & efficiency of output.
* Execution time
* Memory usage

The utility and practicality of the searches will be rated by their:

* Existing functionality.
* Readability.
* Extendability.

## Test Scenario

The scenario will involve an agent moving around a building composed of multiple rooms. Across all tests, the agent will be able to move between rooms and open/close doors. Some tests will have variations of this basic scenario which will include more operations and keywords (e.g. picking up items, locking/unlocking doors, staircases, teleporters...etc).

## Hypothesis

For tests that involve changing size/quantity, we expect the execution time and memory usage to increase as the size/quantity increases. We believe that every time you add a tuple to the start/goal state there will be an exponential increase in time taken. For example, we think that adding 1 tuple to 4 tuples would have a less dramatic impact on the search than adding 1 tuple to 50 tuples.

##### **Goal State Size**

We believe that the more goal operations supplied to ops-search and planner the longer the operation will take to complete, and the more memory it will require. We suspect it will take more time and use more memory as it will have to consider and evaluate more operations.

##### **Tuple Size**

We predict that increasing the data size in each tuple will have little to no effect on speed and efficiency of both ops-search and planner. We assume this because ops-search and planner both look at the first three items in each tuple to perform operations. Any other data entered into each tuple will be ignored by both inference engines.

##### **Tuple Order in State**

Changing the tuple order in the state should have no effect as the state is stored in a set which doesn’t care about order.

##### **Tuple Order in Operations**

Changing the tuple order in the operations list should have an effect on performance if the tuples that exclude most matches are placed closer to the top of the list.

##### **Operator List Size**

We suspect that the more operators we add to the operators collection the longer both inference engines will take to complete the tasks because it will have more operations to consider and apply at each node.

##### **Quantity of Keywords**

With ops-search and planner each tuple starts with a keyword and then two pieces of data. For example, “(locked ?door false)” which translates to “door isn't locked”. The keyword in this tuple is “locked”. On its own, we think that the quantity of different keywords used should not affect the overall speed and efficiency of both ops-search and planner. We do think that having more keywords would make the states and operators harder to read and maintain while also increasing the amount of necessary operators.

##### **Testing Machine**

We plan to run both inference engines on Linux and Windows machines. We assume there will be no differences between the two systems. If we run the systems on machines with higher specs we assume the inference engines will perform better than on lower quality machines. This may be hard to accurately test if the tested machines are not of similar specifications.

##### **State Size**

We believe the larger the state size we specify and provide to both inference engines the more inefficient both inference engines will be. With larger state sizes we think planner will outperform ops-search because it applies additional checks before applying the operator unlike ops-search. Smaller states ops-search will outperform planner as it wont apply the extra validation checks before hand.

##### **Separating World and State**

With boths ops-search and planner a world can be given which states which tells both ops-search and planner thats that will not change.

We predict that specifying a world state will make increase the efficiency of both inference engines as it will have less to consider as those facts cannot change.

Currently, as we are awaiting material related to the planner, we have not yet hypothesised much that would compare the planner against search ops.

## Extra Challenge - Custom Search

Ops-search and Planner work well for generic search solutions that can apply to many scenarios. However, as they aren’t built with an understanding of the problem, they are prone to searching along irrelevant paths.

We aim to make a simple algorithm that figures out how to get an agent from a starting room to an end room. Our belief is that the custom search will be able to outperform the planner and ops-search as the scenarios get larger. However, we also think that the custom search will be harder to understand and modify than ops-search and planner.

The basic version of the custom search would work with an agent, rooms, and doors connecting the rooms. The following steps will be performed:

1. Find all the valid paths from the starting room to the end room, without going to a room that has already been visited. The agent doesn’t move yet so any obstacles such as closed doors are ignored.
2. Count the number of closed doors along each valid path. (Opening a door and moving rooms are both worth 1 move).
3. Using the amount of rooms and closed doors along each path, evaluate which path would be completed in the least moves.
4. Apply the necessary operators to open doors and move rooms to the goal room.

A more difficult version of the custom search would introduce keys and other obstacles that the agent has to interact with to reach the end goal. The first step would be the same as the basic custom search above, but afterwards the algorithm would have to discover what is required to move from room to room.

This adds a lot of complexity as the agent might need to get a key that isn’t along the set path to retrieve a key needed for the path. The path would need recalculating if the agent goes to a room to pick up an object. The agent could simply go back to the starting room each time, but this would almost never find the most efficient path.

If we get the time to program this custom search, we aim to compare it against the planner and ops-search.

## Resources

Useful papers by Simon Lynch for understanding the matcher and its offered functionality. Most of this will be similar across the ops-search and planner.

<https://research.tees.ac.uk/ws/files/4269909/608485.pdf>

<https://www.scm.tees.ac.uk/isg/website/pubs/ELS-2017-Lynch(1).pdf>

##### **Paper by Simon Lynch & Saul Johnson about Clojure and AI:**

<https://www.cognesence.co.uk/downloads/Clojure_Tools_for_Practical_Artificial_Intelligence.pdf>

Details that “The ops-search implementation used in this paper[5] provides a simple, partially optimized implementation of a breadth-first search mechanism for applying simple STRIPS-style operators”. Proof that the ops-search uses breadth-first search.

##### **Breadth -first heuristic search:**

<https://www.sciencedirect.com/science/article/pii/S0004370205002158>

States that it stores all generated nodes of a search graph in memory. Using an open list to store nodes on the search frontier and a closed list to store already-expanded nodes. As its well known, this creates a memory bottleneck that severely limits the scalability of these graph-search algorithms.

##### **Misconceptions on the efficiency of algorithms:**

<https://www.sciencedirect.com/science/article/pii/S0360131503000848>

A lot of our hypotheses involve the theory that larger scenarios/more data will result in poorer efficiency. Some statements in this paper may provide reason if our hypotheses are wrong.

##### Benchmarking:

<https://www.sciencedirect.com/science/article/pii/S0950705111000335>

Defines processes for deciding what benchmarks should be used to understand and evaluate performance.

##### Other

Breadth-first search:

<https://www.sciencedirect.com/science/article/pii/S0743731513001135>

Breadth-first search:

<https://doaj.org/article/2ae3f6a0f75f4eceac1d03248ef0a8f4>

Breadth-first search

<https://www.hindawi.com/journals/sp/2013/702694/abs/>

Rapid Benchmarking for Semantic Web Knowledge Base Systems

<https://drive.google.com/open?id=1U9YjuLdmCBkueV0Tkns-FObhyWtbyqT8>

## Test Distribution

(Craig) Goal state size, Tuple size.

(Josh) Operators list size, Quantity of keywords.

(Oliver) State size, Separating World and State, Duplicate data/Large Pre Ops Size.

(Josh & Craig) Order of tuples in operators + order of operators

* Multiple routes to an end goal. Will the inference engines find the optimal path?

## Extra Notes

Clojure can use Java to discover memory usage.

Could you try to confuse the system? (e.g. having multiple rooms of the same name)?

Can we change the world state during a simulation?

What if you split up the goal state into multiple smaller operations? (E.g. in call 1 you tell ops-search to get a specific key, in call 2 you tell ops-search to go to the end room).

Planner notes

Add visited tags to visited rooms. Planner WILL terminate once it reaches a dead end and try another path