### Base state size

* These tests consisted of changing the state size. Starting from a size of 14 facts going up to 52 facts in increments of two. Each of the tests were run 3 times on each state and gathering the average to see how the change affected the time taken to complete.
* In conclusion in the ops search we found that as the state time grew the time exponentially grew with it. This is the case because with the state size growing in a breadth first search this causes an increased amount of branches on each level of the tree. However when carrying out this test we found that if the facts did not apply to any of the operations then this had minimal effect on the time taken to complete.
* On the planner we found that increasing the state size had no effect on the time taken to complete, this is due to the fact this is a depth first search which is backward chaining. This means that the search will only complete the operations in the post conditions with the specific states needed. So any extra states will not be taken into consideration.

### Optimal Route

-tested if planner and ops-search will find the optimal route when pickup up objects in different rooms

- Planner can’t guarantee an optimal solution as it picks up the items in the order stated in the goal states (holds R key dog cat). Planner would move and pick up the key, then move all the way back and pick up the dog and cat.

-Ops-search can guarantee an optimal route when the objects in the goal state are stated in individual tuples’( (holds R key) (holds R dog) (holds R cat)). Ops-search would pick up the dog, then the cat then the move and pick up the key

-Ops-search with one goal state tuple ‘(holds R key dog cat) would not be able to find the optimal route.

-This is because of the way the pickup operator adds items to the holds tuple. The pickup operator would only add to the end which means they had to be picked up in the specified order.  
-Ops-search with individual tuples would try every combination at each depth so the optimal route is guaranteed.

-Planner is limited in functionality, as even if it could specify multiple tuples in the goal state it would have to pick them up in the order provided as planner is a stack based inference engines.

### Tuple-Order

* We changed the tuple order in the :pre condition for each operator to see if there were any noticeable performance differences.
* With the move, close, open, lock and unlock we noticed a huge difference in time complexity with changes in the order
* With pickup and drop we noticed less of an effect in time complexity.
* Putting the tuple most specific to that operator nearer the top would have a large effect on performance, meaning ops-search could achieve the goal quicker.
* For example, if we moved (in ?agent ?room1) near the top of the :pre for the move operator it could find the goal quicker. This is because it figures out which agent it is, and which room they are in, therefore it eliminates all other rooms and agents being matched to these matcher variables.
* This process is called entropy reduction, which involves placing the conditional logical that eliminates the most results first. (Need to find more about this).
* The order we specified in our tests may not always be the most efficient, as it depends on the state definition. If we put 100 agents in the state and only a few rooms, the order of the :pre tuples may need to be changed for efficiency.
* Planner also can be more efficient with the order change, however this is in the :when clause and not the :pre. There was less of an effect with planner due to the limited state size and operator tuple list.
* We think that Planner is also harder to test this for because planner will already match some of the matcher variables in the :achieves key. This could reduce the amount of times that the operator is checked against for matches drastically.

### Ops-List-Size

* This test looked at changing the amount of operators used when applying ops-search to a state. Operation lists of 5 varying sizes were created and tested against 7 different states.
* We believed that more operators would equate to more time taken to solve the problem.
* Tests 1,2, and 4 show that having more operators being applied to a state, does increase the time taken to solve the problem. This occurs even if the additional operators are not being used.
* Tests 3 and 5 show that the time complexity is affected the most when additional operators are being applied correctly to the state. This is because each valid match of those operators adds a new path to be explored by ops-search.
* Tests 6 and 7 show what happens when operator lists include operators that might lead to more optimal solutions. These tests clearly highlight that the depth reached before solving the problem had just as heavy an impact on the time complexity as having more operators did.

### Alt Planner & Protected Planner

* Protected planner - show them what it looks like and explain the problems
* Protected planer, will get stuck in a loop trying the same path again and again.  
  Josh’s version attempted to fix the problem further, but a new issue surfaced. Instead of creating an infinite loop at a dead end, it would say the agent is now in the dead end room and go backwards claiming a successful solution.
* Alt Planner - Works well but only moving the agent to a room and can be difficult scalability wise. Limits goal state to only move the agent, can't specify opening etc.

### Guard testing

* We tested the use of guard on operations to avoid useless paths being followed (e.g. agent moves from room A to room A).
* For the Move and Teleport tests with ops search, guard was found to be inefficient in smaller scenarios, but became more important as the scenario got larger. This is true especially if the guard operation is placed well in the tuple order.
* The Throw operator testing revealed a scenario where the guard massively reduced memory usage and run times. For example, the larger state test would cause a stack overflow if no guard was specified.
* An alternate Move operation was created and tested with both planner and ops search. In these tests we found that specifying a guard just increased processing time for planner, likely because planner will only perform fruitful operations by looking at the :achieves key.
* In the alternative Move operation tests, ops-search had Stack Overflow issues. The medium tests indicated that the guard was also able to help avoid a Stack Overflow issue but in the larger scenario the exception still occurred.

### Tuple-Size

* We increased the amount of items in a single tuple to see if it affected the performance of ops-search and planer.
* The more items in each tuple, the longer the search took.
* This was because when agent is holding more keys, they have more operations to consider as they could drop them all to begin with.
* When planner has to pick up all the items it gets to 16 operations and breaks. The limit can be increased to get around this.

### Goal-State-Size

* For goal state size testing we increased the number of operations ops-search needed to achieve in the goal state. We started off with a basic state and added one goal to the goal state each time. Next, we created a more advanced scenario to see if the results changed based on the state.  
  Finally, we also checked that if the order of the goal state operations affected the time complexity and if contradictions affected the ability to complete the search.
* We noticed that the more goal states were added the longer it took to complete the search as more depths had to be explored. However, it is not directly proportional to the number of goal states but instead the ultimate depth that had to be reached to achieve all the goal states.
* The order of the goal states did not affect the time complexity as ops-search will find them in the optimal order
* When we specified the the agent had to hold a key and drop the key it was unable to solve the problem, as this is a contradiction.
* When we specified 5 unrelated goal states it was unable to solve the problem as it had to explore to a depth of ?

### Seperate world

* These tests showed the difference between having one large state and splitting the state between that and a world. The world consisted of everything which was a fact and would not change even when operations were applied to the state.
* When running these tests we found that separating the state into state and world did cause the time taken to complete to decrease slightly. This is due to ??? [talk about S. Lynch’s documentation]
* This cannot be applied to planner as it doesn’t have the functionality implemented to add a world.

### Custom Search

* We believe that a custom search would be better as it would be tailored for the scenario and not generic like ops-search and planner.
* It would be able to find a solution quicker as it could drill down through the state and operations list and only apply the once it needed to, to fix the
* Example pseudo-code:
  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.

### Keyword Quantity

In this context, we are referring to a keyword as the first word in a tuple (such as ‘room’). The aim of this test was to see if the number of unique keywords in tuples had an affect on the system performance.

Some simple testing was done to change the ‘unlocked’ and ‘opened’ tuples to transition between ‘unlocked -> locked’ and ‘ opened -> closed’, but no conclusive performance difference could be measured.

We believe that there would be no performance difference based on keyword quantity, because keywords do not have to be stored in memory as definitions. This means that, regardless of the number of unique keywords, the matcher would just match the keywords.

This test was discontinued as it required too many variables to be a valid test. Adding more keywords required more operators which would make it too difficult to determine which change was having the largest effect on the time complexity.

General -

* Ops-search is breadth search
* Planner is depth search
* Ops-search is forward chaining and get the optimal solution
* Planner is backwards chaining and wont always get the optimal solution.
* Planner isnt scaliable and can be challenging to use comparied to ops-search
* Planner is hard to use with connected path scenraios as it can easily get stuck in loops
* Planner is more efficent than ops-search in large scenarios
* Ops-search and planner are roughly the same in small scenarios.