STRIPS PLANNER

Submitted By:

Anannya Upasana(109968397), Satishkumaar Manivannan (109798275), Isha Khanna(109771959)

Problem Statement:

Given a planning domain definition, design a STRIPS planner that will output a sequence of actions for moving from initial state to goal state.

Development Overview:

1. Designed a forward progression planner for simple block and cargo world.
2. Designed a regression planner for the block and air cargo domain.
3. Modified the regression planner to solve two variations of the air cargo domain.
4. Modified A* search and planning graph to accommodate our problem domains.
5. Benchmarked different planners with various problem statements.

Description:

1. Progression and regression:

While implementing the two basic planners, we took care of the loop detection. In forward progression, the variable Sofar is acting as an accumulator to keep track of the actions that have already been applied. In the regression planner, the predicate preserves(Action,Goals) checks that the Action should not delete any Goals with its delete list.
After implementing the basic planners we compared the performance for progression and regression planner for the same domain. While progression works faster for simple small problems regression outweighs progression by a big margin as soon as we start adding more clauses(both redundant and useful). While some of the problems in regression took less than a minute, progression took more than 5 min to come up with a plan. This is due to the huge

branching factor that comes into picture while forward progression. The execution times for the two planners using different test cases are tabulated below.

Progression	Regression	Start State	Goal State	Comments
0	0	[clear(2),clear(4),clear(b),clear(c), on(a,1),on(b,3),on(c,a)]	[on(a,b)]	Smallest test case
1ms	92ms	[clear(2),clear(4),clear(b),clear(c), on(a,1),on(b,3),on(c,a)]	[clear(1),clear(4),clear(3), clear(a), on(c,2),on(b,c),on(a,b)]	
2151ms	557ms	[clear(2),clear(4),clear(b),clear(c), on(a,1),on(b,3),on(c,a)]	[clear(a), on(c,2),on(b,c),on(a,b)]	Removed unecessary clear from final state
>100000ms	250ms	[clear(2),clear(4),clear(b),clear(c), on(a,1),on(b,3),on(c,a)]	[on(c,2),on(b,c),on(a,b)]	Removed unecessary clear from final state
>200000ms	50653ms	[clear(2),clear(5),clear(a),clear(c), clear(d), on(b,1), on(a,b), on(c,3), on(d,4)]	[on(a,2),on(b,5),on(c,4),on(d,a)]	Added an additonal d block and space

Apart from designing the basic progression and regression planners for the aircargo and block world we implemented 3 different scenarios on these domains, which are relevant to real world problems. These are explained with respect to the aircargo domain but are extensible to block world problems without any considerable effort.

Air Cargo problem variation 1: Choosing Planes with maximum fuel efficiency:

One of the variants to the planner that was implemented was to choose a plane to transfer some specified cargo to a particular destination whose fuel consumption rate was the minimum amongst all available planes. The planner also takes into account if the plane has enough fuel to travel from a source to a destination airport. Thus, planes that cannot transfer some cargo because of insufficient fuel are discarded and only planes that can bear the brunt of travelling some particular distance are selected and if the number of such planes are more than one, then the plane with the minimum fuel consumption is selected by the planner.

The predicate plane takes 3 arguments: name of the plane, its fuel consumption rate and the total fuel capacity it has at a particular instance of time. This predicate is maintained dynamically to update the total fuel capacity every time after that particular plane is chosen to travel to a new destination.

The planner was tested for various scenarios or test cases that have been enumerated in the file testbench_variation1.pl.

Observations from the test bench: The planner works well for a limited number of planes, cargoes and airports. However, for a situation where the plane with the minimum fuel consumption rate and the cargo to be transferred are not at the same airport, the planner still chooses that plane, flies it to the airport where the cargo is kept and then transfers the cargo to its destination airport. A more practical approach to deal with this situation would have been to use some plane that is at the same airport as the cargo initially. However, since the planner always chooses a plane with minimum fuel consumption, it does not implement the practical solution in the above case.

The execution times for this variation using different test cases are tabulated below.

Execution Time	Start State	Goal State	Comments
4ms	[at(p1,jfk),at(p2,jfk),at(c1,jfk)]	[at(c1,sfo)]	2 planes that both have capacity to travel to destination airport but only the plane with better fuel efficiency is chosen
9ms	[at(p1,jfk),at(p2,sfo),at(c1,jfk)]	[at(c1,a3)]	p1 does not have capacity to travel from jfk to a3 while p2 does; so p2 goes from sfo to jfk, loads cargo and then delivers it to a3
90 ms	[at(p1,jfk),at(c1,jfk),at(c2,jfk)]	[at(c1,sfo),at(c2,a3)]	plane can carry cargo to 1st airport but not enough fuel to travel to 2nd airport
95 ms	[at(p1,jfk),at(c1,jfk),at(c2,jfk)]	[at(c1,sfo),at(c2,a3)]	1 plane that has capacity to carry both cargoes to their respective destinations
2252ms	[at(p1,sfo),at(p2,jfk),at(c1,jfk),at(c2,sfo)]	[at(c1,a3),at(c2,jfk)]	p1 has enough fuel to transfer c2 from sfo to jfk and p2 has enough to transfer c1 from jfk to sfo

Air Cargo problem variation 2: Handling Planes with limited weight carrying capacity:

A common real world constraint that we have is that any plane can carry only a limited amount of weight in order to ensure optimized efficiency and safety. We have implemented this constraint in our STRIPS planner.

In the cargo domain, a dynamic predicate is maintained for each plane, which saves the total load (in weight) in the plane at any particular moment. One additional precondition for a

cargo to be loaded in the plane is if the plane has left over capacity to accommodate the weight of the cargo.

The planner was tested for different test cases. The test cases with the respective execution times can be found in the testbench_problem3.pl.

Observations from the test bench: Adding this heuristic does not have large impact on the performance for small examples. For more complicated plane problem, the performance is considerably impacted. However, few observations brought out the fact that applying this constraint to the standard regression planner can sometimes reduce the execution time. In this case, some of the planes are rejected in preconditions step only due to lack of capacity, which reduces the overall branching factor.

The same problem solution can be extended to the blocks world where a box can take up only a limited weight over itself.

The execution times for this variation using different test cases are tabulated below.

Execution Time	Start State	Goal State	Comments
66ms	[at(p1, jfk), at(c1, jfk), at(c2,jfk)]	[at(c1, sfo), at(c2,atl)]	2 different destinations
48642ms	[at(p1, jfk), at(p2,jfk), at(c1, jfk), at(c2,jfk), at(c3,jfk)]	[at(c1, sfo), at(c2,sfo),at(c3,atl)]	Added another Airport and cargo for that airport
2524ms	[at(p1, jfk),at(p2, jfk),at(p3, jfk),at(c1, jfk), at(c2, jfk), at(c3, jfk)]	[at(c1, sfo), at(c2,sfo), at(c3,sfo)]	p1->30, p2->20, p3->10, c1->10, c2->10, c3->10
15203ms	[at(p1, jfk),at(p2, jfk),at(p3, jfk),at(c1, jfk), at(c2, jfk), at(c3, jfk)]	[at(c1, sfo), at(c2,sfo), at(c3,sfo)]	p1->20, p2->20, p3->10, c1->10, c2->10, c3->10(reduced p1's capacity so p2 is also used now)
1017675ms	[at(p1, jfk),at(p2, jfk),at(p3, jfk),at(c1, jfk), at(c2, jfk), at(c3, jfk)]	[at(c1, sfo), at(c2,sfo), at(c3,atl)]	p1->10, p2->10, p3->10, c1->10, c2->10, c3->10(Reduced capacity of each plane to 10 so all three are used)

4.1. Planning as search:

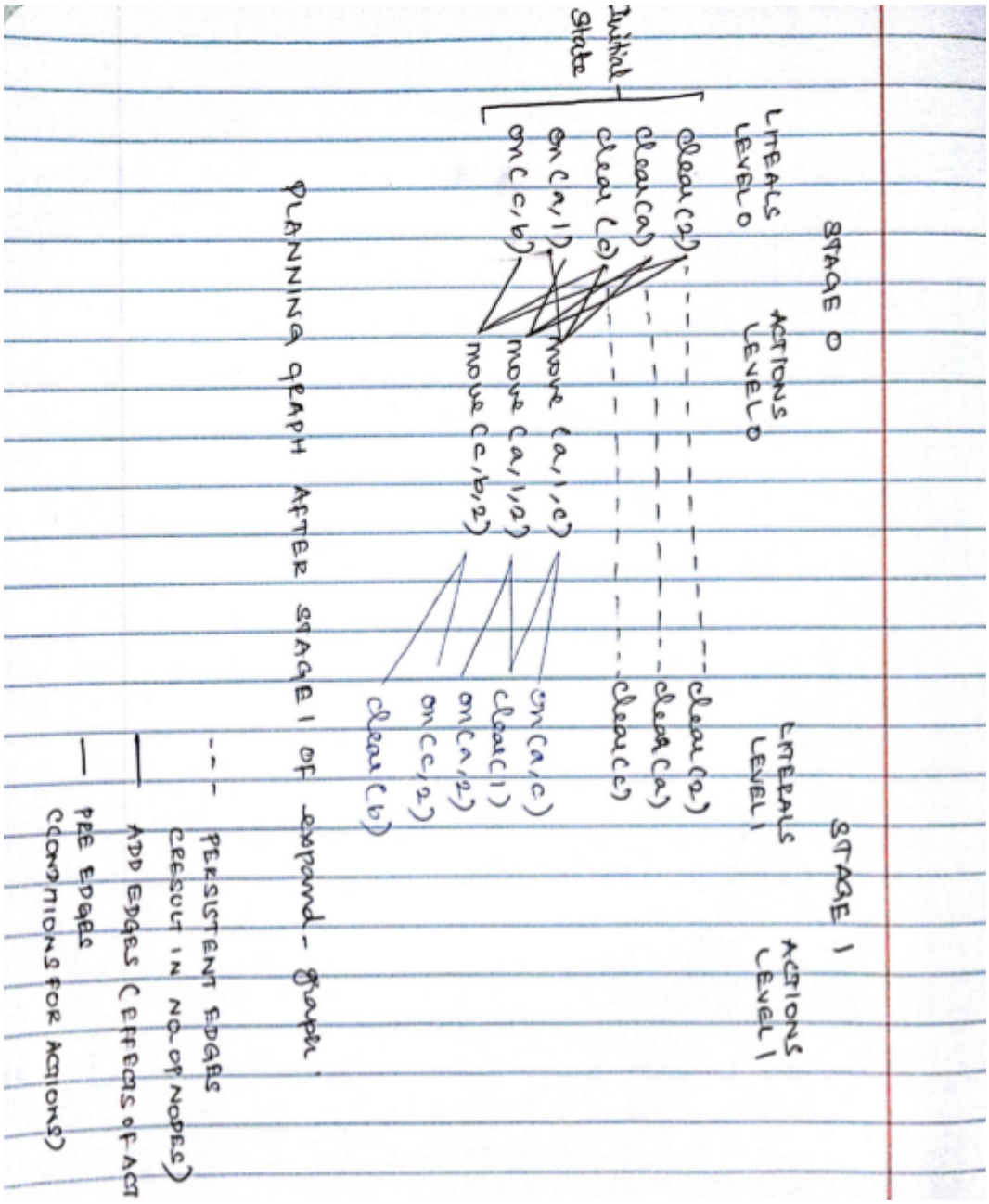
We adapted the code from the textbook for the A* search and defined a variation of the air cargo domain to include distance as cost of the fly action. This implementation of the search planner returned quicker in general than our previous planners. The heuristic function we defined is to check the number of unsatisfied goals remaining by selecting an action. This

helps when we have a set of actions to expand on and one of it adds a goal to the achieve list.

4.2 Graph plan:

We modified an implementation of the planning graph by Blum and Furst found at https://github.com/Mortimerp9/Prolog-Graphplan. The search_graph function in the planning graph gave us the least lenght plan among all the planners because of the nature it achieves goals in parallel.

Below is an illustration of a two level planning graph.



At every stage, the graph_plan, expands the graph by one stage and checks if the literals in the current level satisfy the given goals. If it does, it calls the search_graph to extract the plan from the planning graph each stage at an instance. If not all goals are satisfied, the planning graph is extended on more stage and continues to search recursively.

Graph plan performed the best for the general aircargo domain and the block world domain. Below are the extra test cases we ran that other planners mentioned weren't able to solve.

Initial State	Goal State	Output plan
[at(p1, jfk), at(p2, jfk), at(p3, sfo), at(p4, jfk), at(c1, jfk), at(c2, sfo), at(c3, maa)]	[at(c3, jfk), at(c2, maa), at(c1, sfo)]	[fly(p1,jfk,maa),load(c2,p3,sfo),load(c1,p2,jfk),fly(p2,jfk,sfo),fly(p3,sfo,maa),load(c3,p1,maa),fly(p1,maa,jfk),unload(c2,p3,maa),unload(c1,p2,sfo),unload(c3,p1,jfk)]
[at(p1, jfk), at(c1, jfk), at(c2, sfo), at(c3, maa), at(c4, tky)]	[at(c1, sfo), at(c2, jfk), at(c3, tky), at(c4, jfk)]	> 30 mins. No output
[at(p1, jfk), at(p2, jfk), at(c1, jfk), at(c2, sfo), at(c3, maa), at(c4, tky)]	[at(c1, sfo), at(c2, jfk), at(c3, tky), at(c4, jfk)]	[load(c1,p2,jfk),fly(p1,jfk,sfo),load(c2,p1,sfo),fly(p2,jfk,sfo),unload(c1,p2,sfo),fly(p1,sfo,tky),unload(c2,p1,tky)]

Files Submitted:

Domain Definitions
aircargo_domain.pl : Domain definition for aircargo problem
block_domain.pl : Domain definition for block world problem
Progression
progression_planner.pl : Simple Planner using Progression
Regression
regression_planner.pl : Simple Planner using Regression
Variation 1
regressionplanner_variation1.pl: Regression Planner for air cargo domain choosing fuel efficient planes
testbench_variation1.pl: Different test cases for testing the planner in checkf.pl

Variation2
aircargo_variation2.pl: Aircargo Domain definition for planes with limited weight carrying capacity
regression_variation2.pl: Regression Planner used for applying limited weight carrying capacity constraint
testbench_variation2.pl: Different test cases for testing the heuristic3 planner.

graphplan folder contains the planner and the domain files used to test.
Syntax: plan(InitialState, Goals, Plan)
bestfirst folder contains the planner and the domain files used to test the A* search.

Resources:

1. Prolog Programming for Artificial Intelligence, Ivan Bratko
2. Fast Planning Through Planning Graph Analysis, Avrim L. Blum
Merrick L. Fürst
3. A Heuristic Estimator for Means-Ends Analysis in Planning, Drew McDermott
4. Artificial Intelligence, Peter Norvig, Stuart Russel.