This report compares the efficiency, speed and optimality of solutions to three planning problems when solved using a number of different non-heuristic and heuristic search types. The three planning problems are summarized below:

```
Init(At(C1, SFO) ∧ At(C2, JFK)
        ∧ At(P1, SFO) ∧ At(P2, JFK)
        ∧ Cargo(C1) ∧ Cargo(C2)
        ∧ Plane(P1) ∧ Plane(P2)
        ∧ Airport(JFK) ∧ Airport(SFO))
Goal(At(C1, JFK) ∧ At(C2, SFO))
```

Illustration 1: The initial state and the goal state of problem 1, defined in Planning Domain Definition Language. Plane 1 and cargo 1 at airport SFO, plane 2 and cargo 2 at airport JFK. The goal is to have cargo 1 at airport JFK and cargo 2 at airport SFO.

```
Init(At(C1, SFO) ∧ At(C2, JFK) ∧ At(C3, ATL)
        ∧ At(P1, SFO) ∧ At(P2, JFK) ∧ At(P3, ATL)
        ∧ Cargo(C1) ∧ Cargo(C2) ∧ Cargo(C3)
        ∧ Plane(P1) ∧ Plane(P2) ∧ Plane(P3)
        ∧ Airport(JFK) ∧ Airport(SFO) ∧ Airport(ATL))
Goal(At(C1, JFK) ∧ At(C2, SFO) ∧ At(C3, SFO))
```

Illustration 2: The initial state and the goal state of problem 2, defined in Planning Domain Definition Language. Plane 1 and cargo 1 at airport SFO, plane 2 and cargo 2 at airport JFK and plane 3 and cargo 3 at airport ATL. The goal is to have cargo 1 at airport JFK and cargos 2 and 3 at airport SFO.

```
Init(At(C1, SFO) ∧ At(C2, JFK) ∧ At(C3, ATL) ∧ At(C4, ORD)
        ∧ At(P1, SFO) ∧ At(P2, JFK)
        ∧ Cargo(C1) ∧ Cargo(C2) ∧ Cargo(C3) ∧ Cargo(C4)
        ∧ Plane(P1) ∧ Plane(P2)
        ∧ Airport(JFK) ∧ Airport(SFO) ∧ Airport(ATL) ∧ Airport(ORD))
Goal(At(C1, JFK) ∧ At(C3, JFK) ∧ At(C2, SFO) ∧ At(C4, SFO))
```

Illustration 3: The initial state and the goal state of problem 3, defined in Planning Domain Definition Language. Plane 1 and cargo 1 at airport SFO, plane 2 and cargo 2 at airport JFK, cargo 3 at airport ATL and cargo 4 at airport ORD. The goal is to have cargos 1 and 3 at airport JFK and cargos 2 and 4 at airport SFO.

The following search types have been explored to identify a solution to each problem: breadth first search, breadth first tree search, depth first graph search, depth limited search, uniform cost search, recursive best first search with mock heuristic h1, greedy best first graph search with mock heuristic h1, a* search with mock heuristic h1, a* search with h_ignore_preconditions heuristic and a* star search with h_pg_levelsum heuristic. The performance of each one is summarized in Table 1 below. Table 2 below shows the minimum length solutions to each problem found by each type of search. The performance of non-heuristic search types (breadth first search, breadth first tree search, depth first graph search, depth limited search and uniform cost search) and a* star searches with different heuristics will be discussed in more detail.

		Problem 1	Problem 2	Problem 3
1. breadth_first_search	expansions	43	3343	14663
	goal tests	56	4609	18098
	new nodes	180	30509	129631
	time (ms)	101.5719740026	30883.3864859989	190215.226184999
2. breadth_first_tree_search	expansions	1458	timeout	timeout
	goal tests	1459	timeout	timeout
	new nodes	5960	timeout	timeout
	time (ms)	1669.4957500004	timeout	timeout
3. depth_first_graph_search	expansions	12	582	627
	goal tests	13	583	628
	new nodes	48	5211	5176
	time (ms)	16.1982579957	6105.5552289981	6056.7730510011
4. depth_limited_search	expansions	101	222719	timeout
	goal tests	271	2053741	timeout
	new nodes	414	2054119	timeout
	time (ms)	174.2527620008	1786390.043616	timeout
5. uniform_cost_search	expansions	55	4853	18151
	goal tests	57	4855	18153
	new nodes	224	44041	159038
	time (ms)	74.1778069932	23491.2074369931	129146.454381
6. recursive_best_first_search h_1	expansions	4229	timeout	timeout
	goal tests	4230	timeout	timeout
	new nodes	17029	timeout	timeout
	time (ms)	4739.8833070001	timeout	timeout
7. greedy_best_first_graph_search h_1	expansions	7	998	5398
	goal tests	9	1000	5400
	new nodes	28	8982	47665
	time (ms)	9.3188909996	4585.6415140006	27372.2914749997
8. astar_search h_1	expansions	55	4853	18151
	goal tests	57	4855	18153
	new nodes	224	44041	159038
	time (ms)	64.3014910002	22009.1203979991	93960.3113869998
9. astar_search h_ignore_preconditions	expansions	41	1450	5038
	goal tests	43	1452	5040
	new nodes	170	13303	44926
	time (ms)	50.5424129988	6817.5467970013	27026.8005710004
10. astar_search h_pg_levelsum	expansions	11	86	314
	goal tests	13	88	316
	new nodes	50	841	2894
	time (ms)	759.2659459988	55288.7530360022	274644.236332999

Table 1: Performance of a number of search types for the three problems. Values in the red background indicate outcomes where a non-optimal solution was found. Timeout values indicate that search was terminated because it was not finished after running for more than 20 minutes.

	Problem 1	Problem 2	Problem 3

1. breadth_first_search	Load(C2, P2, JFK)
	Load(C1, P1, SFO)
	Fly(P2, JFK, SFO)
	Unload(C2, P2, SFO)
	Fly(P1, SFO, JFK)
	Unload(C1, P1, JFK)
		Load(C2, P2, JFK)
		Load(C1, P1, SFO)
		Load(C3, P3, ATL)
		Fly(P2, JFK, SFO)
		Unload(C2, P2, SFO)
		Fly(P1, SFO, JFK)
		Unload(C1, P1, JFK)
		Fly(P3, ATL, SFO)
		Unload(C3, P3, SFO)
			Load(C2, P2, JFK)
			Load(C1, P1, SFO)
			Fly(P2, JFK, ORD)
			Load(C4, P2, ORD)
			Fly(P1, SFO, ATL)
			Load(C3, P1, ATL)
			Fly(P1, ATL, JFK)
			Unload(C1, P1, JFK)
			Unload(C3, P1, JFK)
			Fly(P2, ORD, SFO)
			Unload(C2, P2, SFO)
			Unload(C4, P2, SFO)

2. breadth_first_tree_search	Load(C2, P2, JFK)
	Load(C1, P1, SFO)
	Fly(P2, JFK, SFO)
	Unload(C2, P2, SFO)
	Fly(P1, SFO, JFK)
	Unload(C1, P1, JFK)
		timeout	timeout

3. depth_first_graph_search	N/A	N/A	N/A
4. depth_limited_search	N/A	N/A	timeout

5. uniform_cost_search	Load(C1, P1, SFO)
	Load(C2, P2, JFK)
	Fly(P1, SFO, JFK)
	Fly(P2, JFK, SFO)
	Unload(C1, P1, JFK)
	Unload(C2, P2, SFO)
		Load(C1, P1, SFO)
		Load(C2, P2, JFK)
		Load(C3, P3, ATL)
		Fly(P1, SFO, JFK)
		Fly(P2, JFK, SFO)
		Fly(P3, ATL, SFO)
		Unload(C1, P1, JFK)
		Unload(C2, P2, SFO)
		Unload(C3, P3, SFO)
			Load(C1, P1, SFO)
			Load(C2, P2, JFK)
			Fly(P1, SFO, ATL)
			Load(C3, P1, ATL)
			Fly(P2, JFK, ORD)
			Load(C4, P2, ORD)
			Fly(P2, ORD, SFO)
			Fly(P1, ATL, JFK)
			Unload(C1, P1, JFK)
			Unload(C2, P2, SFO)
			Unload(C3, P1, JFK)
			Unload(C4, P2, SFO)

6. recursive_best_first_search h_1	Load(C1, P1, SFO)
	Load(C2, P2, JFK)
	Fly(P2, JFK, SFO)
	Unload(C2, P2, SFO)
	Fly(P1, SFO, JFK)
	Unload(C1, P1, JFK)
		timeout	timeout

7. greedy_best_first_graph_search h_1	Load(C1, P1, SFO)
	Load(C2, P2, JFK)
	Fly(P1, SFO, JFK)
	Fly(P2, JFK, SFO)
	Unload(C1, P1, JFK)
	Unload(C2, P2, SFO)
		N/A	N/A

8. astar_search h_1	Load(C1, P1, SFO)
	Load(C2, P2, JFK)
	Fly(P1, SFO, JFK)
	Fly(P2, JFK, SFO)
	Unload(C1, P1, JFK)
	Unload(C2, P2, SFO)
		Load(C1, P1, SFO)
		Load(C2, P2, JFK)
		Load(C3, P3, ATL)
		Fly(P1, SFO, JFK)
		Fly(P2, JFK, SFO)
		Fly(P3, ATL, SFO)
		Unload(C1, P1, JFK)
		Unload(C2, P2, SFO)
		Unload(C3, P3, SFO)
			Load(C1, P1, SFO)
			Load(C2, P2, JFK)
			Fly(P1, SFO, ATL)
			Load(C3, P1, ATL)
			Fly(P2, JFK, ORD)
			Load(C4, P2, ORD)
			Fly(P2, ORD, SFO)
			Fly(P1, ATL, JFK)
			Unload(C1, P1, JFK)
			Unload(C2, P2, SFO)
			Unload(C3, P1, JFK)
			Unload(C4, P2, SFO)

9. astar_search h_ignore_preconditions	Load(C1, P1, SFO)
	Fly(P1, SFO, JFK)
	Unload(C1, P1, JFK)
	Load(C2, P2, JFK)
	Fly(P2, JFK, SFO)
	Unload(C2, P2, SFO)
		Load(C1, P1, SFO)
		Fly(P1, SFO, JFK)
		Unload(C1, P1, JFK)
		Load(C2, P2, JFK)
		Fly(P2, JFK, SFO)
		Unload(C2, P2, SFO)
		Load(C3, P3, ATL)
		Fly(P3, ATL, SFO)
		Unload(C3, P3, SFO)
			Load(C1, P1, SFO)
			Fly(P1, SFO, ATL)
			Load(C3, P1, ATL)
			Fly(P1, ATL, JFK)
			Unload(C1, P1, JFK)
			Load(C2, P2, JFK)
			Fly(P2, JFK, ORD)
			Load(C4, P2, ORD)
			Fly(P2, ORD, SFO)
			Unload(C2, P2, SFO)
			Unload(C3, P1, JFK)
			Unload(C4, P2, SFO)

10. astar_search h_pg_levelsum	Load(C1, P1, SFO)
	Fly(P1, SFO, JFK)
	Load(C2, P2, JFK)
	Fly(P2, JFK, SFO)
	Unload(C1, P1, JFK)
	Unload(C2, P2, SFO)
		Load(C1, P1, SFO)
		Fly(P1, SFO, JFK)
		Load(C2, P2, JFK)
		Fly(P2, JFK, SFO)
		Load(C3, P3, ATL)
		Fly(P3, ATL, SFO)
		Unload(C1, P1, JFK)
		Unload(C2, P2, SFO)
		Unload(C3, P3, SFO)
			Load(C2, P2, JFK)
			Fly(P2, JFK, ORD)
			Load(C4, P2, ORD)
			Fly(P2, ORD, SFO)
			Load(C1, P1, SFO)
			Fly(P1, SFO, ATL)
			Load(C3, P1, ATL)
			Fly(P1, ATL, JFK)
			Unload(C1, P1, JFK)
			Unload(C2, P2, SFO)
			Unload(C3, P1, JFK)
			Unload(C4, P2, SFO)

Table 2: Minimum length solutions found by each search type. N/A values indicate that the solution found was not minimum length. Timeout values indicate that search was terminated before returning a solution because it was running for more than 20 minutes. Each background colour indicates a different solution for each problem (i.e. although the solution length is the same, the order of action execution to achieve the goal is different)

Non-heuristic search

Non-heuristic, or uninformed, search iterates through problem nodes until a solution is found. It does not reaccess each node as search progresses to guide the search towards a solution. The paragraphs below discuss and compare the performance of breadth first, depth first and uniform cost search algorithms in solving the three logistical planning problems described earlier. Figures 4, 5 and 6 below visually illustrate their performance on the three problems, respectively. Note, that for each problem, only the outcomes of searches that did not time out are plotted.

In breadth first search, all the possible options for each node are explored until the solution (the list of actions to the goal state) is found – i.e. the states after taking all the possible actions that can be taken from the initial state are explored, then for each of these states, the states after taking all the possible actions that can be taken are explored and so on. Because all options are being explored, breadth first search is complete (it will always find the shallowest goal node in the search tree) and optimal (it will find the most optimal solution) if the solution optimality for the problem is directly proportional to the path cost. However, because the whole search tree is explored, this search is expensive in terms of time and space required. If every state has *b* successors and the solution is at search tree level *d*, the time and space complexity is $O(b^d)$[1] – there is an exponential growth in the time and space required to solve the problem. The performance is even poorer if no record of already explored nodes is kept and the search continuously revisits the explored options which is the case during tree search. Graph search keeps track of nodes already visited and does not revisit them. For the three problems described here, breadth first tree search timed out for the two more complicated problems, problem 2 and problem 3, and it had the second worst performance out of the ten search types explored in terms of time taken, number of expansions, goal tests and new nodes for solving problem 1 (see Table 1). Breadth first graph search (named just 'breadth first search') performed better - it identified solutions for all the three problems in under 20 minutes. A particularly interesting point to note is that it performed better time-wise than a* search with the complex h_pg_levelsum heuristic. This goes to show that sometimes a 'dumb' uninformed search that explores numerous unnecessary nodes can be a better choice than a 'smart' heuristic one with a complex heuristic that significantly reduces the number of expansions, goal tests and new nodes.

In depth first search, each branch of the search tree is explored to greatest possible depth before the next branch is explored. The search terminates when a solution is found. As a result, the depth first search in not complete (the solution found is not necessarily the shallowest) and not optimal. However, the search space complexity is only $O(bm)$ where *m* is the maximum search depth and *b* is the number of successors each state has (the branch factor). It is worth noting that if *m* is very large (approaching infinity), the search would just keep exploring down a single path of the first expanded node, never searching through the rest of the tree. Hence, in practical applications, depth first search is usually used with a limit *l* for search depth – nodes at depth *l* are treated as if they have no successors.[2] This way, a greater breadth of the tree is explored. For the three problems explored here, both the depth first graph search without a limit and depth limited search with *l* set to 50 were used to find a solution. As expected, both find non-optimal solutions with extremely long path lengths and hence can not compete with the breadth first search in this sense. However, depth first graph search without a limit, in the case where it is allowed to keep going down a single path of the first explored node, is significantly faster in finding a solution than breadth first search, and, as expected, it executes fewer expansions and goal tests and explores fewer nodes than breadth first search. Hence, depth first search might be a good choice for problems where solution path length is not of great importance, but for the three problems analysed here it is not of great use.

Finally, the uniform cost search always expands the node with the lowest path cost, rather than expanding one with the shortest path. In these particular problems, there were no weights assigned to different nodes (e.g. distances between airports or fuel consumption), so uniform cost search had

1 *p.82 Artificial Intelligence: A Modern Approach* by Norvig and Russell, 3rd edition

2 *p.87, Artificial Intelligence: A Modern Approach* by Norvig and Russell, 3rd edition

performance comparable to breadth first graph search, expanding fewer more nodes since it does not stop when a solution is found but when it has explored all the nodes at the goal's depth to make sure the solution found is the minimal cost one. For the particular problems explored here, there is no significant advantage for using uniform cost search over the breadth first search.

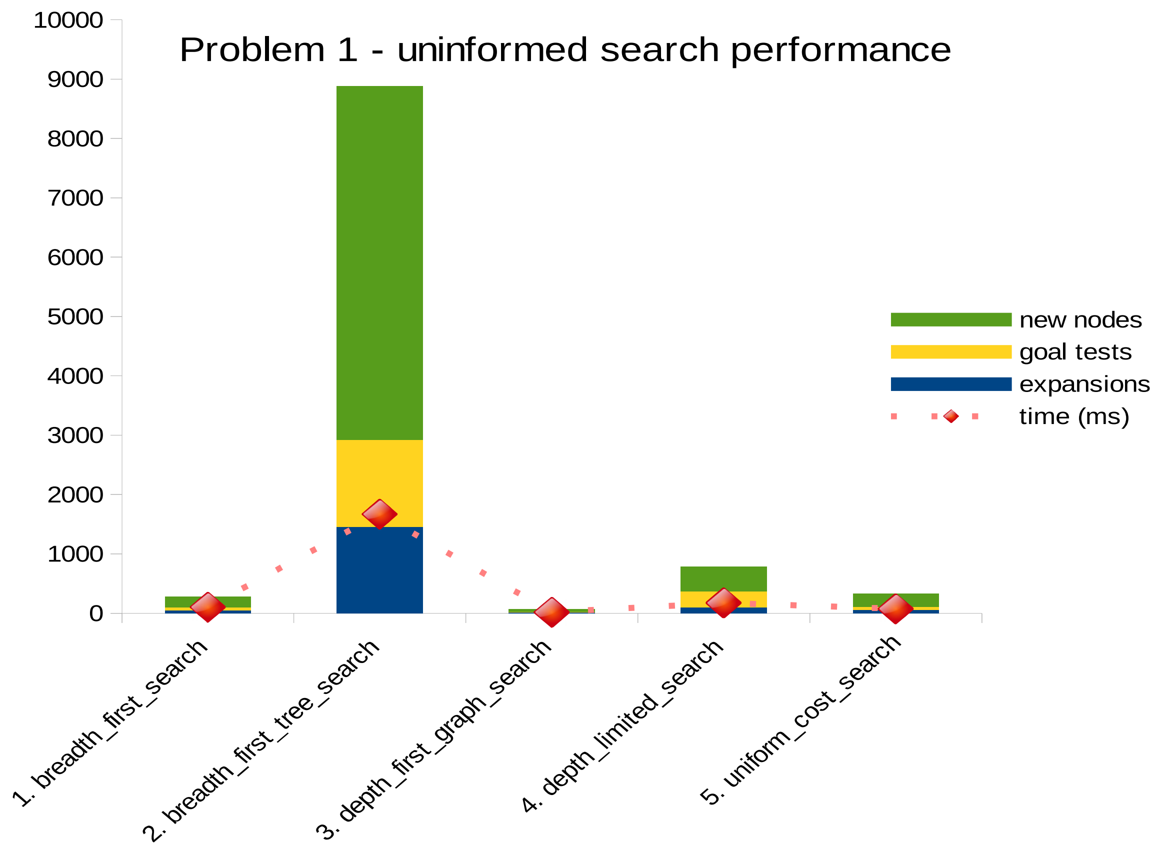

Illustration 4: Performance of uninformed search types in problem 1

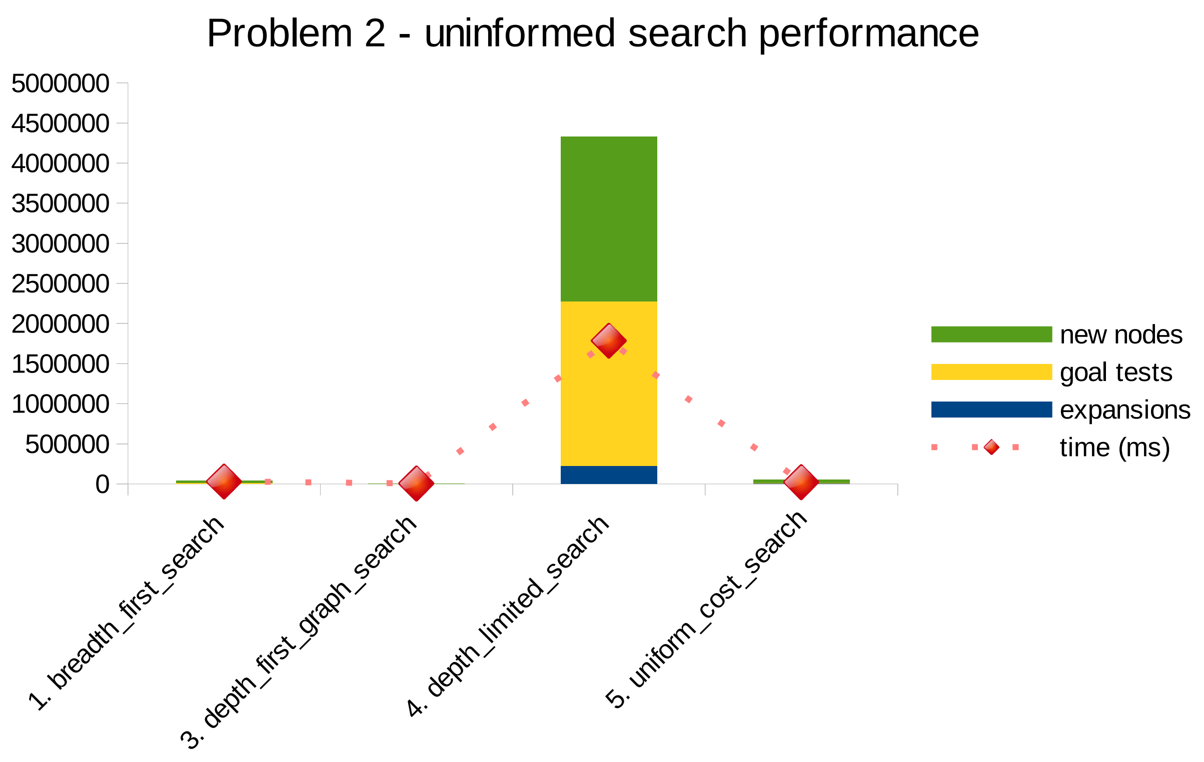

Illustration 5: Performance of uninformed search types in problem 2

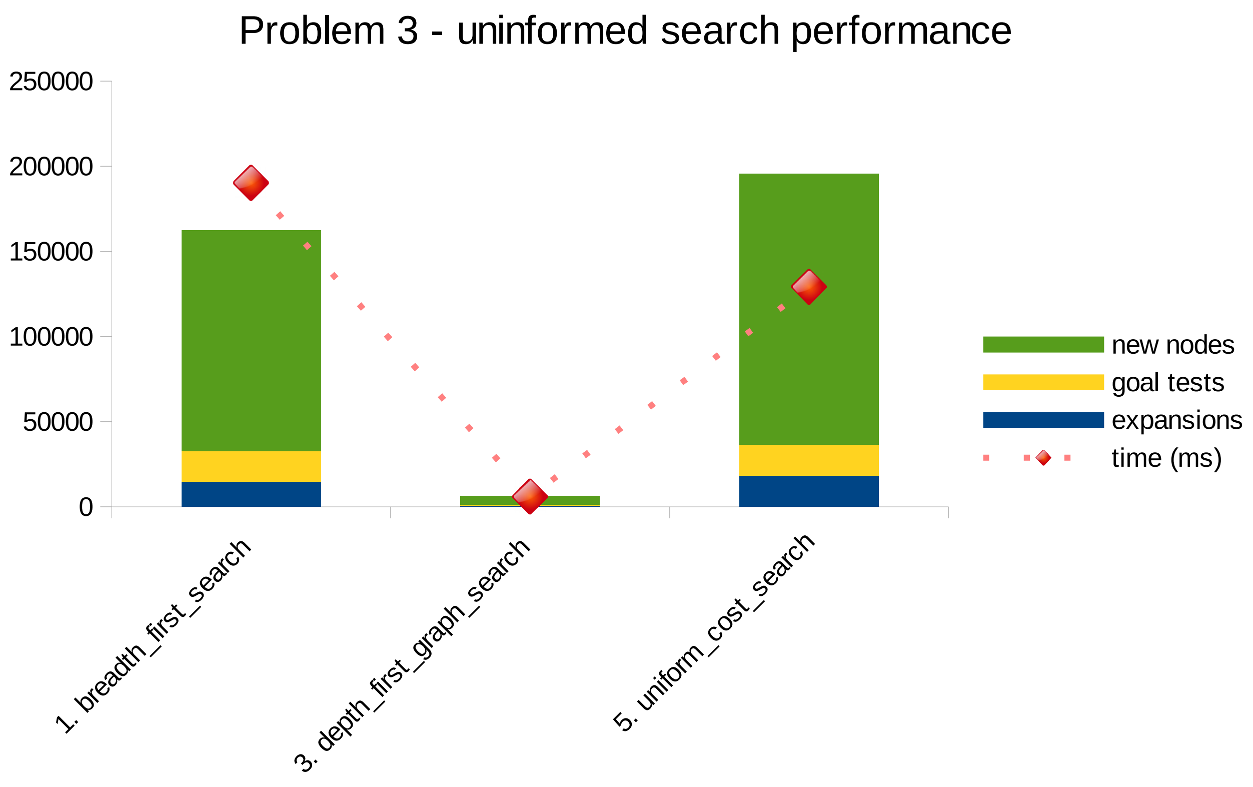

Illustration 6: Performance of uninformed search types in problem 3

A* heuristc search

Heuristic search uses problem specific knowledge to assess each problem node during the search process. As a result fewer nodes need to be explored to find a solution. The flip side is that if the heuristic function is overly complex, the ‘smart’ search might take longer than the non-heuristic search to arrive at a solution. Ideally, one wants to find a simple but effective heuristic to drive the search.
A* search in particular is a very popular heuristic search algorithm, used to tackle numerous problems. A* search combines the lowest path cost estimate used in non-heuristic uniform cost search with a problem-specific heuristic to guide the selection of nodes for solving the problem. Hence, if we use a mock heuristic, it’s performance is pretty much identical to that of uniform cost search (explore the results of uniform cost search and A* star search with h1 heuristic in Tables 1 and 2). The true power of A* search is revealed when a relevant heuristic is used. Two different heuristics were investigated here: h_ignore_preconditions and h_pg_levelsum. The former simply removed the restraints of the problem turning it into a more relaxed problem, i.e. for each node state it counted the number of minimum moves required to achieve the goal state if the preconditions of each action are ignored. This is a simple heuristic, requiring minimal computation and it was particularly effective in solving the logistical planning problems – looking at the results for solution to the third problem, it has reduced the number of expansions, goal tests and new nodes, and the time required to solve the problem more than 3-fold in comparison to A* star search with mock h1 heuristic. In fact, it has given the bests results out of all ten search types explored – it finds an optimal solution and it does it in a very short time comparing to other search types that return an optimal solution. The other heuristic, h_pg_levelsum is a complex heuristic – it generates a planning graph for each node (state and action layers leading up to a solution) and adds up the minimum layer depths where each goal conditional is first found. This is very effective in further reducing the number of expansions, goal tests and new nodes – for the third problem, more than staggering 15 times in comparison to A* search with h_ignore_preconditions heuristic – but due to the time required to generate a planning graph for each node to be explored, the time performance is around 10 times worse than that of A* search with h_ignore_preconditions heuristic. Hence, it can be argued that for the problems described here, it is not necessary to devise a heuristic as complex as h_pg_levelsum to tackle them. Figures 7, 8 and 9 below visually summarize the performance of A* search with the free different heuristics on the three problems.

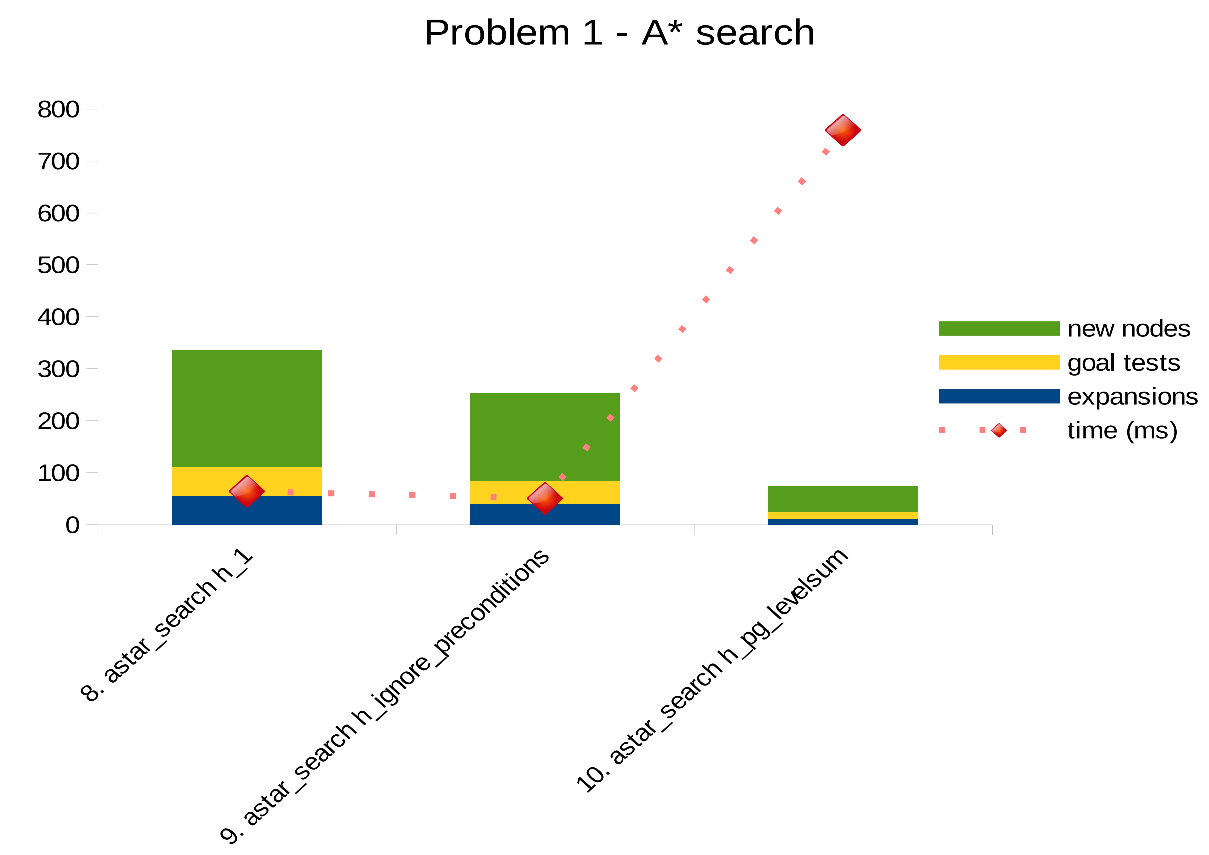

Illustration 7: Performance of A search with different heuristics on problem 1*

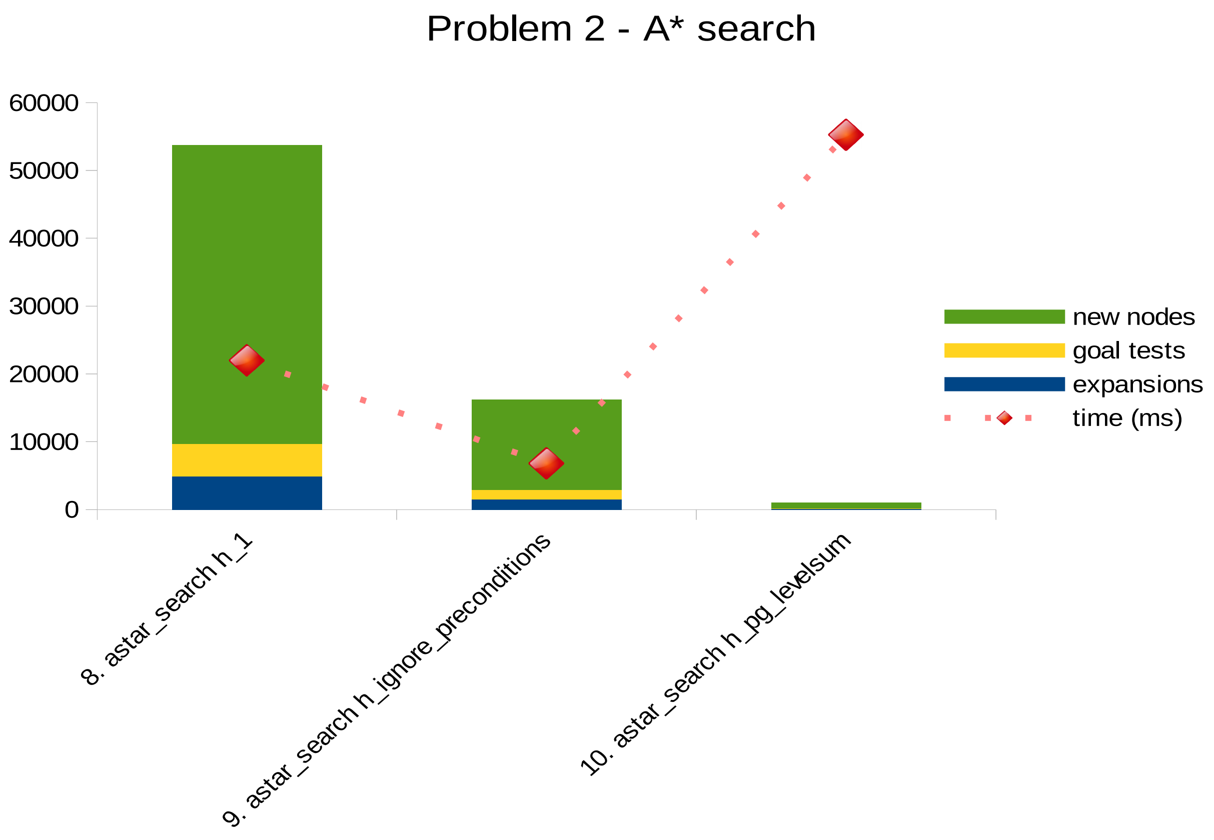

Illustration 8: Performance of A search with different heuristics on problem 2*

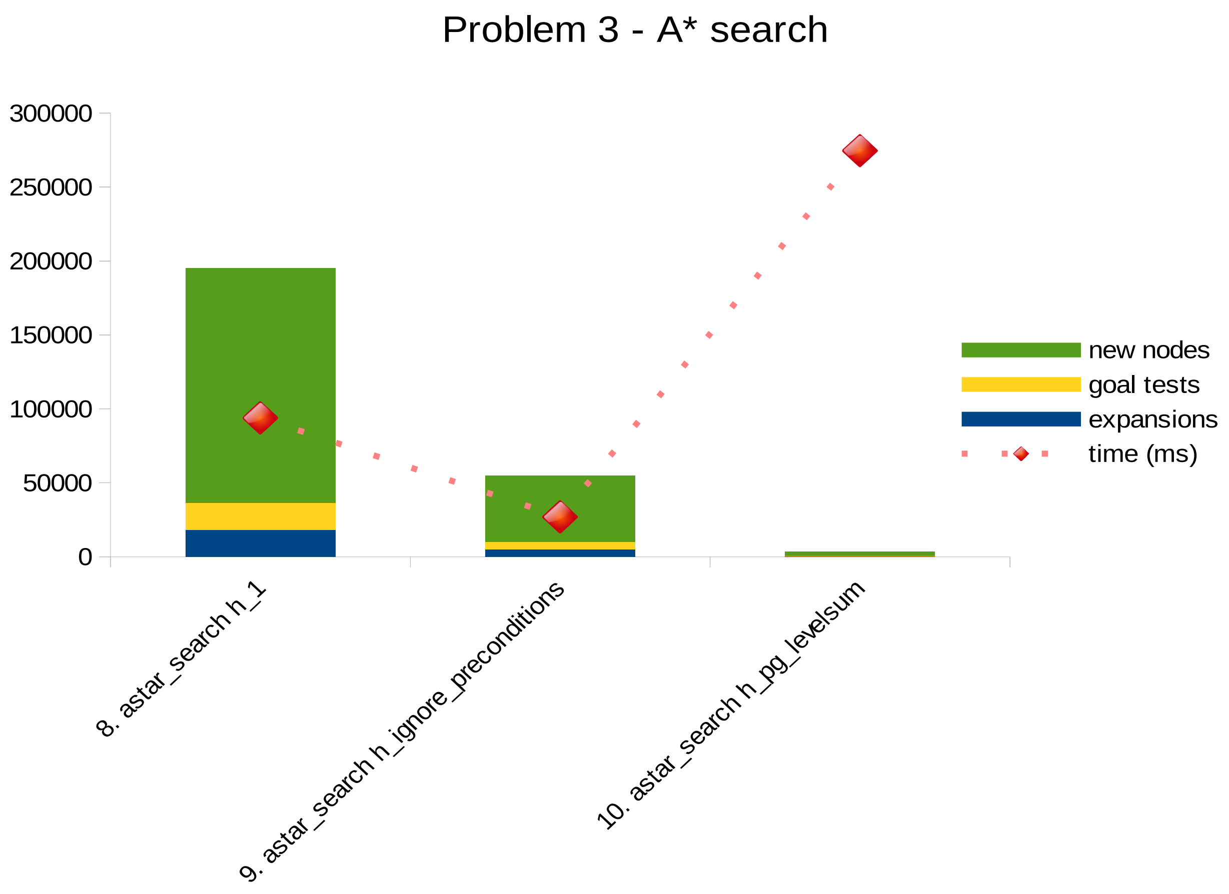

Illustration 9: Performance of A search with different heuristics on problem 3*