Artificial intelligence planning has had many applications over the years. The techniques in this realm have been used in classical planning, control theory, and theorem proving.[5] There have obviously been numerous advancements, both big and small, that have gotten us to this point. In this research review, we focus on three such advancements: the `STRIPS` algorithm, the `GRAPHPLAN` algorithm, and optimizations to `GRAPHPLAN`.

The first big advancement in this area was the `STRIPS` algorithm. `STRIPS` was developed as planning algorithm for a mobile robot that could navigate and push objects around in a multi-room environment[3]. Although this was just one part of the "Skakey the robot" project, it established the foundation of classical planning terminology. For example, all the propositional logic we used in the project was originally used in the `STRIPS` algorithm[6]. However, this algorithm had limitations: it assumed that actions could be applied one at any time, that nothing changed except as a result of the actions, and that actions were instantaneous[3]. Of course these assumptions are not a realistic model of the world we occupy and it was shown that `STRIPS` could not solve some relatively simple problems[5]. This led to some other advancement that eventually lead us to `GRAPHPLAN`.

`GRAPHPLAN` is celebrated because it is a simple algorithm and is orders of magnitude faster than its predecessors[6]. This is essentially the algorithm covered in the lecture videos, however we did not cover very much (if any) of the "solution-extraction phase." To the best that I can tell, we didn't code any of that phase in the project either. We did implement several parts of the "graph-expansion phase." One of the main reasons that `GRAPHPLAN` is so efficient is the mutual-exclusion (or mutex) relations. Part of our project was to code the various ways that action and literal nodes could be mutex. The goal with graph expansion is to get to the point that all goal literals appear at a level, all being pairwise not mutex. Then the next step follows.

The solution-extraction phase of `GRAPHPLAN` is a follows. First, at level i, we try to find a non-mutex set of actions that achieves all the desired goals using a depth-first search. If we find a set of non-mutex actions that give us all the goals at level i, then we do the same thing at level i-1 with our "goals" now being the preconditions of the set of actions we found. When `GRAPHPLAN` gets to level 0, then we only check to see if the preconditions we have are in the initial state. If at any level we are unable to find a set of non-mutex actions, then we continue expanding the graph (from [5] and [6])

Now there many optimizations we can make to `GRAPHPLAN`. First, we note that graph expansion is at worst polynomial time, while solution-extraction is exponential[1]. However, the majority of the computation time is taken up in graph expansion, therefore any optimizations made to graph expansion are valuable[6].

One such optimization was done in the project: the closed-world assumption. This the idea that any proposition not explicitly given as true in the initial state can be assumed to be negative. Another optimization which we have encountered in this Nanodegree Program is constraint satisfaction. It is straightforward to see the connection between constraint satisfaction problems and the solution-extraction phase of `GRAPHPLAN`[6].

The last optimization that I will mention here is in-place graph expansion. In [6], we see that literal and action levels are monotonically increasing, while mutexes at each level are monotonically decreasing. Using this information, we can get rid of the multilevel graph expansion approach in GRAPHPLAN and simply two a bipartite with action nodes on one side and literal nodes on the other. An arc from a literal node to an action node indicates a precondition and an arc from an action node to a literal node indicates an effect. We can then use markers to keep track of mutex conditions that we update as we consider more and more actions. This would speed up our algorithm because we would have few nodes and only need to create them once.

It is easy to see how these technologies build on each other: STRIPS spawned ideas like GRAPHPLAN, and GRAPHPLAN, with some of the optimizations mention here, spawned new techniques like IPP[4] and SGP[2]. Hopefully a participant in this program will make the next big move in planning graph search.

References

[1] Avrim L. Blum and Merrick L. Furst. Fast planning through planning graph analysis. *Artificial Intelligence*, 90, 1997.

[2] David E. Smith Corin R. Anderson and Daniel S. Weld. Conditional effects in GRAPHPLAN. *In Proceedings of the Fourth International Conference on Artificial Intelligence Planning Systems*, 1998.

[3] Richard E. Fikes and Nils J. Nilsson. Strips, a retrospective. *Artificial Intelligence*, 59, 1993.

[4] Jörg Hoffmann Jana Koehler, Bernhard Nebel and Yannis Dimopoulus. Extending planning graphs to an adl subset. *n Proceedings of the Fourth European Conference on Planning*, 1997.

[5] Stuart Russel and Peter Norvig. *Artificial Intelligence: A Modern Approach.* Pearson Education, Inc., 3rds edition, 2010.

[6] Daniel S. Weld. Recent advances in ai planning. *AI Magazine*, 20(2), 1999.