Scribe notes from Monday, Feb. 18 2002: GraphPlan

Amy Davis

February 26, 2002

1 An Overview of GraphPlan

The goal of GraphPlan is to generate plans that are represented in a graph of a given length or shorter. Because the graph structure is not a tree, we don't need to be concerned about branching. The structure of GraphPlan may be large, but remains polynomial. Using this structure, we represent an exponential number of possible solutions in a polynomial size and a polynomial time building process.

Even though building this planning graph is polynomial, searching through the planning graph remains exponential. However, they seek to minimize the size of each search within the planning graph. A cut-off mechanism functions like Forward-Checking in the search tree, by marking future actions that are mutually exclusive to actions already taken. If all actions possible for reaching a goal are marked, then the goal becomes unreachable (or cut-off), and backtracking must take place.

All actions in the planning graph are fully instantiated. There remain no variables. It generates a Partially ordered Plan (POP), like UCPOP, but unlike Prodigy, which is a linear planner. This POP is of optimal (shortest) length, though length as a criterion for quality is now questioned. When searching to find this POP, GraphPlan has a low sensitivity to the order of the goals since search considers minimal sets of actions at the action levels. This allows for less recursion, less backtracking, and less sensitivity to variable ordering.

If at any particular level in the search, a particulor combination of propositions is found unsolvable, this subset of goals is *memoized*, and GraphPlan backtracks to the previous level. The authors propose and student four factors that improve the performance of GraphPlan:

1. Mutual exclusion constraints. These are propagated through the graph prior to search.
2. Parallel plans. These allow for shorter paths and a smaller structure.
3. Memoization, which allows us to avoid making the same mistake in search twice.
4. Building first, then searching. This keeps the size of search small.

In section 6, we see some other improvements that can be made to GraphPlan. These include:

- Computing a value n. This is the number of goals that cannot be realized simultaneously. Therefore a graph must be at least n steps long to reach them. The problem is analogous to finding maximal cliques, which is difficult to approximate. In problems like the Traveling Salesman, computing this value n makes GraphPlan much faster.
- Regression Analysis. We can remove non-useful preconditions from the initial condition by regressing from the goal to find relevant preconditions. This is done by collecting all preconditions and add-lists of all the actions and goals. Then we mark the ones that are either in the goal, or in preconditions to

actions that may be needed. An action may be needed if it has add-list members that are in the goal or preconditions to actions already selected. This allows us to find preconditions that are definitely not relevant in linear time.

- Learning from plans—this has lots of potential, and Etzioni is a key player in this domain.
- Symmetry exploitation. This has been tried by Maria Fox and Derek Long [Fox and Long 1998] and Rintanen (on invariants) [Rintanen 1997]. However symmetries such as interchangeabilities have not been exploited.
- Bi-directional search: searching from the beginning is $\mathcal{O}(b^d)$. Searching from both the beginning and the end allows d to be halved. However, it seems not very widely used in practice.
- The inclusion of other constraints. We don't know of other work doing this.
- Max-Flow algorithms. These generally don't perform well with backward chaining, as in the GraphPlan algorithm. However they may allow us to consider actions with different costs.

GraphPlan is limited by the fact that no new objects can be created. All objects must exist at the beginning of the plan. Additionally, only pairwise mutex constraints are employed by GraphPlan. GraphPlan may benefit from including other kinds of constraints—specifically capacity constraints.

2 Explanations

A minimal set of actions is a set in which every action is necessary to meet the goal set. GraphPlan restricts its attention to minimal sets of actions in order to restrict possible backtracking. Using a set of actions that are larger than the minimal set could actually consider actions that require a larger set of preconditions, and are thus harder to satisfy.

On page 8, the proof of Theorem 1 includes n, m, p, l, and t. However, the t is not shown in the proof. t is the number of levels in the graph, and the $\mathcal{O}$s given are for each level of the graph. A re-derivation of the proof is as follows:
n objects
p propositions at S_0
m STRIPS operators
l items in the longest add-list
t levels to the planning graph
k arguments to the highest-arity operator

There are n objects. An operator is instantiated with objects, so each operator can be instantiated n^k ways. Objects can be repeated in the instantiation of the operator, so an arity 5 operator can take $n \times n \times n \times n \times n$ possible instantiations (we understand this thanks to Praveen). Each of these instantiations can create at most l new terms. So the effect of an action is $\mathcal{O}(ln^k)$. At any level, every operation can apply, and there are m operations. Additionally there are a possibility of p no-ops that apply. So the maximum size of a proposition level is $\mathcal{O}(mln^k + p)$. At an action level, each operator can be instantiated n^k ways. There is m operators, so the maximum size of an action level is $\mathcal{O}(mn^k)$.
Each action level is $\mathcal{O}(mn^k)$.
Each proposition level is $\mathcal{O}(mln^k + p)$.
There are t possible levels (of each).
So, the whole thing is $\mathcal{O}(tmln^k + p)$ (since the action level size is smaller than the proposition level size).

Therefore, as the paper states, GraphPlan is polynomial in n, m, p, and l, not in k. k is the arity of the operators. This allows for combinatorial explosion in the possibilities at each level.

In the last paragraph of section 4, we see two instances in which goal ordering has an effect. (We had previously noted that GraphPlan was insensitive to goal ordering). These cases are:

- During the goal-step creation
- When we find a solution

3 Monday's Discussion

Xu Lin While we mentioned in class that a ten-minute to a few-second comparison in running-time is significant whether or not the architectures are different, Xu Lin disagrees. He recalls using C that required 3 seconds to solve a problem that someone's code in Java took 30 minutes. Using the different machine coding language is not acceptable. However, the comparison of how the two strategies increase remains significant. GraphPlan's slope is *much* flatter than the other strategies.

Xu Lin In a planning graph, just because the goals can be achieved does not mean that there exists a valid plan. It may be that all the goal are met by the planning graph, but there is no solution at all. If the search space can be reduced, it may be easier to find valid plans. Technically there's an infinite number of plans, because a plan can loop. GraphPlan avoids this by finding the shortest plan.

Amy GraphPlan is a high-commitment planner because it assigns time-stamps to an action immediately. The paper contradicts the conventional wisdom that a least-commitment planner is the most effective. This actually enables GraphPlan to be faster.

Rob GraphPlan may suffer from scalability problems because the size of the planning graph is $\mathcal{O}(p+mln^k)$ where k is the largest number of formal parameters in any operator. If k is large (operators have high arity), is could cause problems. Indeed, such problems were showed in the SATPlan paper to occur even when k was 4 by lifting.

Cory The idea of "weakly complete" is used for the basic GraphPlan algorithm, which, at each step either discovers a plan or proves that no plan having that many steps or fewer is possible. The ideas of leveling-off and memoization make the algorithm more "complete" in the normal sense.

Cory One cannot judge the "sensible-ness" of a plan—it is quite subjective. However the authors claim to give empirical evidence that a plan is sensible. In the discussion we noted that the length of a plan determines its quality. This would hold for its sensible-ness as well. While shortness was a common criteria, it's now being questioned. One could also argue that the lowest cost would be a good criteria.

Dan The optimizations that they provide were likely discovered through trial and error.

References

Fox, Maria and Long, Derek 1998. The Automatic Inference of State Invariants in TIM. *Journal of Artificial Intelligence Research* 9:367–421.

Rintanen, Jussi 1997. A planning algorithm not based on directional search.