Planning and Factor Graphs

Frank Dellaert

April 14, 2020

1 Planning

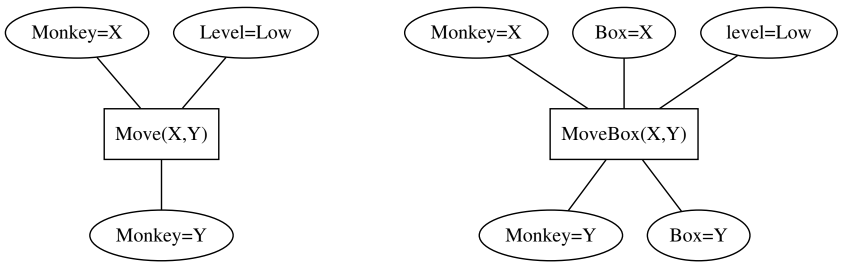

Figure 1: Two actions a monkey can take, along with their preconditions and postconditions. Note that in these diagrams the postconditions only specify the variables that changed.

let us consider a very simple planning problem, taken from the Wikipedia entry on STRIPS, a language to express planning problems in. You can read a blog post about the problem, including a cool photo, in this medium post. However, rather than propositional logic as is common in STRIPS, let us use multi-valued, discrete variables to describe the problem:

1. A monkey can be at locations A,B, or C, denoted by the variable $Monkey \in \{A, B, C\}$
2. Bananas, suspended from the ceiling, are also at *one* of these locations: $Bananas \in \{A, B, C\}$
3. The monkey has the bananas, or not: $HasBananas \in \{True, False\}$
4. Theres is a box that the monkey can move, $Box \in \{A, B, C\}$
5. Finally, the monkey can climb on the box: $Level \in \{Low, High\}$

The monkey is now faced with the following planning problem: *it is at location A, the bananas are at location B, and the box is at location C. The monkey wants to have the bananas.*

Normally, in STRIPS, there is now a lot of textual descriptions of actions, but instead we will use some graphical templates to define actions in terms of preconditions and postconditions, shown in Figure 1 and 2. For example, for the monkey to move from X to Y, or take the **action** $Move(X, Y)$, the precondition is that it is at location X, i.e., $Monkey = X$, and the postcondition is $Monkey = Y$.

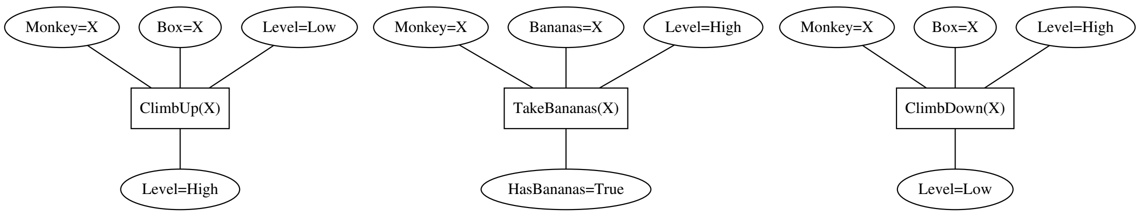

Figure 2: Three more actions the monkey can take, now involving the *Level* of the monkey, which is either *High* or *Low*. The monkey can only have the bananas when it is high :-).

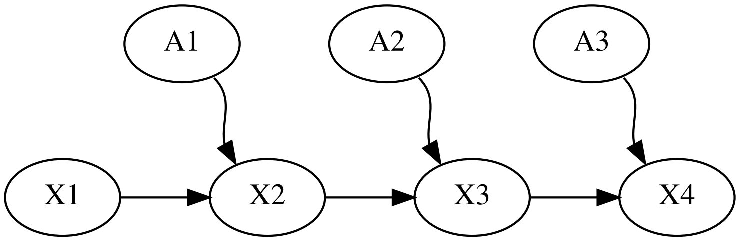

Figure 3: A dynamic Bayesian network (DBN).

2 Dynamic Bayesian Networks

By introducing time and defining a state, we can define something akin to a **dynamic Bayesian network** (DBN) that describes a planning problem, only there are no probabilities involved. If we define the **state** X as $X = (Monkey, Bananas, HasBananas, Box, Level)$, then the initial state can be written as a prior $P(X_1)$, and the effect of taking an action at time t, denoted by A_t, can be written in the form of a conditional distribution, yielding the DBN,

$$P(X_1) \text{ and } P(X_{t+1}|X_t, A_t)$$

a fragment of which is shown in Figure 3.

However, in planning it is common to regard the states not as monolithic but **factored**, and it will be beneficial to express this in the DBN as well, by making explicit for each variable how it depends on the variables at the previous time, and the chosen action. A slice of this factored DBN for the monkey problem is shown in Figure 4. Intuitively speaking, factoring the state into multiple variables rather than one large state variable makes it possible to omit some of the arrows, because the next state of certain variables might not depend on *all* variables in the previous time step.

For deterministic planning, the entries in the conditional probability table will contain either 0 or 1, because the actions have deterministic outcomes. Interestingly, this immediately suggests the possibility of making the outcome of actions probabilistic: an action might or might not succeed, in which case this becomes a Markov decision problem, i.e., a factored MDP [Guestrin et al., 2003].

3 Graphplan

The DBN can be unrolled using a factor graph representation, as shown in Figure 5. This will come down to a factor graphs of **hard constraints**, because there are no probabilities involved. Each constraint factor is happy only if the state X_{t+1} is the outcome of applying action A_t in state X_t. Note that in each time slice, because the actions are parameterized, the actions variable A_t can

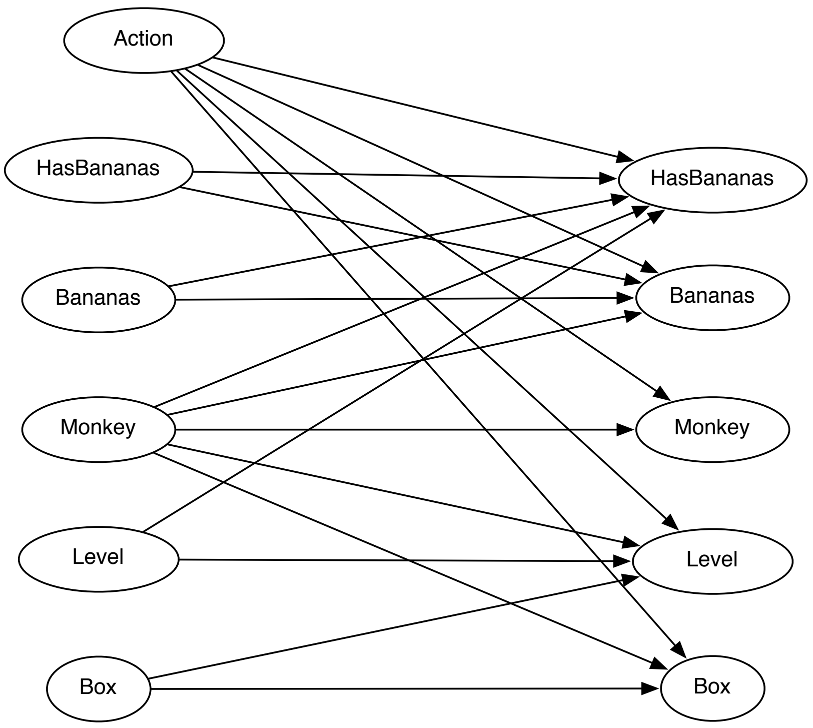

Figure 4: A slice of the *factored* DBN describing the "monkey" planning problem. There is a single conditional distribution per variable, taking into account which action is executed.

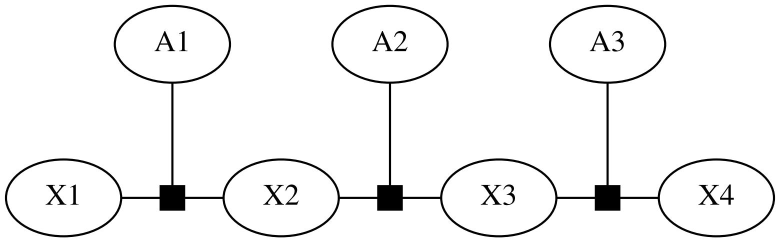

Figure 5: The factor graph representation of a planning DBN.

take on a large number of values. In the monkey domain there are three locations, A,B, and C, and hence 9 possible $Move(X,Y)$ actions. Likewise, there are 9 $MoveBox$ actions, and three $ClimbUp$, $TakeBananas$, and $Climbdown$ actions, for a total of 27 actions in this simple domain.

Graphplan [Blum and Furst, 1997] is a seminal algorithm introduced by Merrick Furst (at Georgia Tech!) and Avrim Blum, who take this one step further: they allow for actions to be taken *in parallel*. In the monkey example, because we have 27 possible actions, we now have 2^{27} possible sets of actions! Clearly, this is a combinatorial explosion that we cannot easily deal with. However, graphplan factors the single action variable from the DBN formulation as well, using 27 **binary** variables, one for each possible action. Doing so does make it easy to reason about the mutual exclusion constraints between action instantiations, which now become binary constraints.

The propositional approach of graphplan is taking it too far for some: rather, we could take a CSP approach and represent obviously mutually exclusive actions with a single, multi-valued discrete variable. In the monkey example, it is clear that the monkey cannot do any of the actions in parallel, and we recover the simpler DBN formulation above. This "bundling" of mutually exclusive actions

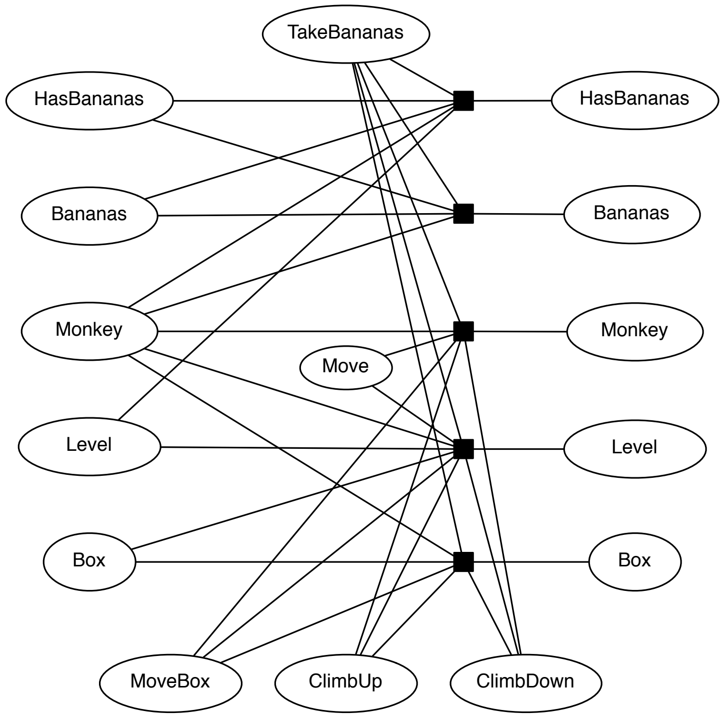

Figure 6: Factor-graph representation of factored planning DBN. There is a factor that governs the transition of every variable, conditioned on the action.

into a single multi-valued variable is often done as a pre-processing step for modern search-based planners [Helmert, 2006, Richter and Westphal, 2010].

One time-slice of a compromise factor graph for the monkey problem is shown in Figure 6. This is not exactly a planning graph as discussed by Blum and Furst, but it comes quite close. In graphplan, the states are encoded using propositional variables, and so are the actions, making for an even finer-grained graph. However, most of the information obtainable using a planning graph is represented in this factor graph slice.

4 Search

Solving the planning problem can now be seen as searching in an infinitely long factor graph. However, we can provide a lower-bound on the time it will take to execute the plan. By instantiating the initial state at time $t = 1$, we can propagate the constraints in time, discovering the first time at which the goal state can hold. If we cannot find a plan of that length, we can extend the graph by 1 and try again, i.e., an **iterative deepening** type strategy. Backward search would start by instantiating the goal state $X_T = G$ at time T, which will automatically lead to a depth first search. Forward search starts with the start state $X_1 = S$, and searches forward in time.

We can use both **constraint propagation** and heuristics to make the search efficient. Search in these spaces can be quite expensive, think of trying to solve a Sudoku using search! Hence, just like in a Sudoku, search is typically combined with forward checking/constraint propagation at every iteration. In a Sudoku this goes as follows: if a certain number is filled into the grid, we erase all other values in all other cells that are incompatible with it. This generalizes easily to the current planning setup.

Good **heuristics** currently are the key to the best performance in planning. Rather than local constraint propagation, they provide a longer term perspective. Indeed, if we remember the analogy with HMMs, we are really trying to do inference in a possible very long chain of states/actions, finding feasible plan that takes us all the way from the start state to the goal state. Brute-force inference can be seen to do this in a slice-by-slice manner. However, if we had a notion for each state X_t of how long it would take us (or how much it would cost us) to plan from that state to the goal state (or a lower bound, i.e., a heuristic), we could explore the most promising states first.

Many modern heuristics are derived from relaxed versions of the original planning problem. For example, the family of Fast Forward (FF) planner [Hoffmann and Nebel, 2001] from the University of Freiburg uses a relaxation of the planning problem in which delete lists are ignored, and is among the most successful planners today. A delete list is the list of pre-conditions that have to be negated: in the monkey example, we should make sure that if the $Move(X, Y)$ action is chosen, the monkey is no longer at X. But if we ignore this basic fact, the monkey can be at *both* X and Y: clearly a fiction, but maybe the monkey can get to the bananas faster this way! FF uses graphplan as a subroutine to find the length of a relaxed plan (without delete lists) which is always an underestimate of the actual optimal plan length.

5 SATplan

SATplan [Kautz and Selman, 1992, Kautz and Selman, 1999] uses the same ideas as above, but uses propositional logic instead of constraints, to turn the problem into one of Boolean satisfiability (SAT). Any constraint satisfaction problem (CSP) as above can be converted into a SAT problem in a variety of ways, and then a SAT solver can be called to solve the planning problem.

It was noted by the authors of MaxPlan [Xing et al., 2006], another planner based on satisfiability, that the iterative deepening approach is faced with the task of repeatedly proving that no plan can be found for successive time steps. This is potentially very expensive, as this implies exhausting the entire search space. Hence, instead MaxPlan uses a non-optimal planner to find the maximum plan length and then successively reduces from that upper bound, which can be done much faster. They tied for first place in that year's planning competition, the other being the 2006 version of SATplan.

This asymmetry between satisfiable and unsatisfiable also shows up in the choice of algorithm used. In [Kautz and Selman, 1996] it is shown that WalkSAT, a greedy random walk version of SATplan, can find plans very quickly for a given time horizon T, *if one exists.* However, proving that a plan is optimal is beyond the capabilities of WalkSAT, which is incomplete: instead, this requires an exhaustive search.

6 Summary

Classical planning can be seen as a "hard" version of Markov decision problems, which can be described using dynamic Bayes nets. In turn, Bayes nets can easily be converted to factor graphs that represent the corresponding planning problems as a constraint satisfaction problem (CSP). The dominant paradigm for optimizing planners is search, and both constraint propagation and heuristics are important to make search efficient. Some of the most efficient solvers convert planning CSPs to Boolean satisfiability (SAT), for which very efficient search algorithms have been developed. But, for easy problems, a simple greedy search like WalkSAT can work very well.

References

[Blum and Furst, 1997] Blum, A. and Furst, M. (1997). Fast planning through planning graph analysis. *Artificial Intelligence*, 90(1-2):281–300.

[Guestrin et al., 2003] Guestrin, C., Koller, D., Parr, R., and Venkataraman, S. (2003). Efficient solution algorithms for factored mdps. *J. of Artificial Intelligence Research*, 19:399–468.

[Helmert, 2006] Helmert, M. (2006). The fast downward planning system. *J. of Artificial Intelligence Research*, 26:191–246.

[Hoffmann and Nebel, 2001] Hoffmann, J. and Nebel, B. (2001). The FF planning system: Fast plan generation through heuristic search. *J. of Artificial Intelligence Research*, 14(1):253–302.

[Kautz and Selman, 1992] Kautz, H. and Selman, B. (1992). Planning as satisfiability. In *Eur. Conf. on AI (ECAI)*, pages 359–363.

[Kautz and Selman, 1996] Kautz, H. and Selman, B. (1996). Pushing the envelope: Planning, propositional logic, and stochastic search. In *Nat. Conf. on Artificial Intelligence (AAAI)*, pages 1194–1201.

[Kautz and Selman, 1999] Kautz, H. and Selman, B. (1999). Unifying SAT-based and graph-based planning. In *Intl. Joint Conf. on AI (IJCAI)*, volume 16, pages 318–325.

[Richter and Westphal, 2010] Richter, S. and Westphal, M. (2010). The LAMA planner: Guiding cost-based anytime planning with landmarks. *J. of Artificial Intelligence Research*, 39:127–177.

[Xing et al., 2006] Xing, Z., Chen, Y., and Zhang, W. (2006). Maxplan: Optimal planning by decomposed satisfiability and backward reduction. In *Proc. Fifth Int'l Planning Competition, Int'l Conf. on Automated Planning and Scheduling (ICAPS'06)*, pages 53–56.