Personal Planning Problem

CMU CS 15-887 Project: Due on Dec 05, 2014 at 5pm

Xiao-Feng Xie (xfxie@alumni.cmu.edu)

1 Motivation

Planning is a notoriously hard combinatorial problem. Domain-independent computational complexity for propositional STRIPS [6] planning and related formalisms are quite well-known [3]. In general, it is already $PSPACE$-complete to determine if a planning instance has any solutions. In the case where some planning domain is fixed in advance [5], planning in a domain that can be represented in the STRIPS subset of PDDL [7] can be $PSPACE$-complete. Further complexity results have been shown for some standard planning domains. For planning domains in a common transportation theme [8], the plan existence problem might be decided in polynomial time, but the bounded plan existence problem is often NP-complete.

Various methods have been proposed to solve planning problems. In plan-space planning, UCPOP is a sound, complete, partial order planner whose step descriptions include universal quantification and conditional effects [20]. In state-space planning, Prodigy [19] is a nonlinear planning architecture that has choice points on either to apply an applicable operator or to continue solve pending goals. In Graphplan [1], a compact structure called a planning graph is constructed to approximate the reachability graph with alternating proposition and action layers. In Satplan [13], the planning problem is formalized in terms of propositional satisfiability, and can them be solved using stochastic search algorithms. In heuristic-search planning (HSP) [2], the heuristic is automatically extracted from the declarative problem representation, and is used in the context of best-first algorithms for guiding the search in solving planning problems. As an improved HSP, the fast-forward (FF) planning system [10] uses a different search technique, i.e., an enforced form of hill-climbing that combines local and systematic search, using relaxed Graphplan to inform the search with a goal distance estimate and with helpful actions. Some advanced structure clues, e.g., landmarks, might be further considered to accelerate the search [15]. By calling FF on a carefully constructed deterministic variant of the planning problem, FF-Replan [21] can perform highly effective in planning under uncertainty. In addition, If reward information is available, uncertainty might be handled using (partially observable) Markov decision processes [12, 11, 16, 17] for obtaining or choosing a policy, i.e., a mapping from states to actions.

Personal planning is of significance for improving personal production and operating efficiency of personal time resource. Most of the current available methods and software which called as "personal planning solutions" are however actually aiming to satisfy or to optimize a personal scheduling rather than personal planning, where a set of tasks will be reordered according to their time sequence and required time amount to achieve them. A generic personal planning problem has certain constraints, a set of preferences, and some expected accumulated reward, which are not

as simple as a personal scheduling problem. In this study, we will present a primary solution to a personal planning problem. To our best knowledge, it is still an open question in practice for solving a complex personal planning problem.

2 Problem Description

For a planning problem involving deterministic actions and complete information, a basic state space consists of a finite set of states S of the world, a finite set of actions A, a state transition function $t : S \times A \to 2^{|S|}$. For more realistic domains, there is also a reward function $r : S \times A \to \Re$ that provides the reward of doing action $a \in A$ in state $s \in S$. In an actual state model, it contains the state space, and additionally a given initial state s_0, and a set of goal states s_G. A *plan* is then a set of actions that generate a state trajectory to transform s_0 to s_G. The solution is optimal when the total cost is minimized.

In practice, the encoding language for planning tasks is usually the Planning Domain Definition Language (PDDL) [7]. The PDDL language provides a foundation for an expressive representation, which enables domain models for realistic applications. PDDL is intended to be a neutral specification of a planning domain, and is designed to provide no domain-specific advice for planners.

Planning is more about "thinking" about the way of modeling real-world domains. In a personal planning problem, the world is spatio-temporal in dimension, and consists of different activities subject to various real-world (e.g., physical, social, and economic) constraints. The underlying spacial domain can be described with an undirected graph $G =< N, L >$, where each node $n \in N$ is a *location* to hold some activities, and each link $l \in L$ describes the connectivity between two locations. Similar to the *Logistics* domain [18], we consider the case of multi-modal transportation with *vehicles*, where an *airplane* can fly between some locations called *airports*, and a *car* can drive between some locations (with solid links, where at least one side is a *city* but not an airport). Some basic activities are considered. First, a person can *load*/*unload* a document or luggage or parcel, which is defined as *package*, to/from a vehicle, e.g., car or airplane, and carry a package from one location to another. Second, a person would like to *meet* some *experts* at different locations.

Let the person be the only agent. The basic domain is defined by *objects*, *predicates*, and *actions*.

The constant *objects* describe entities that exist in a planning domain for the activities of the agent. The requirement "typing" is considered to formalize the property of different entities, and reducing any redundant *predicates* and *actions* that operate on objects of same types. There are following object types:

- Two *location* types, i.e., $\{city, airport\}$: Each location is a node in the underlying graph G.

- One *package* type: A package is located at a location, and can be transported to another location.

- One *expert* type: Each expert lives at a given location. As a task, the agent would like to meet an expert (for some potential gains).

- Two *vehicle* types, i.e., $\{car, airplane\}$: Each vehicle can transport the agent (and a package, if has) from an origin location to a destination location with a link. For each car, at least one of the two locations should not be an airport. For each airplane, both locations should be airports.

- One *target* type, i.e., $\{package, expert, vehicle\}$.

The *predicates* consist of a list of literals that are taken to be true at all times in this domain. There are following predicates:

- (is-airport ?l - location): It examines if the location is an airport.
- (is-connect ?l - location ?l - location): It provides a link between two locations.
- (at-own ?l - location): It indicates if the agent is currently at a given location.
- (at ?t - target ?l - location): It checks if a target is currently at a given location.
- (in ?p - package ?v - vehicle): It indicates if a package is currently in a vehicle.
- (conflict ?e1 - expert ?e2 - expert): It indicates if there is a conflict to visit expert e2 before expert e1.
- (is-meet ?e - expert): It indicates if the agent has successfully met an expert.

The *actions* are operator-schemas with parameters, which are instantiated during execution. There are following actions:

- drive (?v - car ?from - location ?to - location): The agent drives a car from one location to another location with a link. At least one of the two locations is not an airport.
- fly (?v - airplane ?from - airport ?to - airport): The agent flies from one airport to another airport with an airplane.
- load (?v - vehicle ?p - package ?l - location): The agent loads a package into a vehicle at a given location.
- unload (?v - vehicle ?p - package ?l - location): The agent unloads a package from a vehicle at a given location.
- meet (?e - expert, ?l - location): The agent meets an expert at a given location. For any expert that met earlier, the is-meet state is negated if there is a conflict with the current expert.

The real-world personal planning problem will be treated as an online planning problem. At each time point, only the problem in a given horizon is considered and it will be simplified as a deterministic planning problem. For any uncertainty, a carefully constructed deterministic variant of the planning problem might be considered as in FF-Replan. The initial state contains the underlying graph G of the world, the initial location of the agent, packages, experts, and vehicles. The set of goals might contain the final location of the agent and packages, and the status of meeting with experts.

The planning aims to minimize the time cost of finishing all goals. For simplicity, the time spend on each action is assumed to be the same. Hence the cost will be propositional to the total number of actions, since the agent can only do one action at each time.

3 Solving Approach

Various planners have been proposed for deterministic planning problems. Many of them, e.g., HSP, IPP [9], BLACKBOX [14], MIPS [4], Fast Downward [9], however, do not support sub-types that used in this personal planning domain. Only the FF planner is considered in this project.

The FF planner belongs to the class of heuristic-search planning, where a heuristic function is derived from the task specification of each planning instance to guide the search in the state space. The basic system architecture of FF contains two basic components: Enforced Hill-Climbing, and Relaxed Graphplan.

Given a state, relaxed Graphplan is used as a heuristic estimator to estimate the distance to a goal in polynomial time, and additionally selects a set of promising successors to the state, called helpful actions, for pruning the search, although without preserving completeness.

Enforced Hill-Climbing is a forward searching engine to work on each state, and to perform exhaustive breadth-first search for the better states, by assuming state spaces are simple in structure with small-sized local valleys and plateaus. The search is only complete on any task that is dead-end free, although it is $PSPACE$-complete to decide the property of dead-end free for a task.

Two techniques are integrated for handling goal orderings. The added goal deletion is a heuristic to cut out branches when some goal has been achieved too early. The goal agenda approach feeds the goals in the order recognized in a preprocessing phase by looking at all pairs of goals.

4 Evaluation

Figure 1 gives an example of the personal planning domain to be evaluated. There are totally 10 locations ($\mathbf{A}-\mathbf{J}$), in which **F** and **G** are airports that hold an airplane $a1$ to connect two parts $\mathbf{A}-\mathbf{F}$ and $\mathbf{G}-\mathbf{J}$. Initially, the agent is located at **A**. There are two vehicles, where $v1$ and $v2$ are respectively located at **A** and **G**, and they can be respectively seen as a private car and a rental car. There is one package $p1$, which is initially located at **B**. There are four experts, $e1-e4$, whom are respectively located at **C**, **D**, **J**, and **E**.

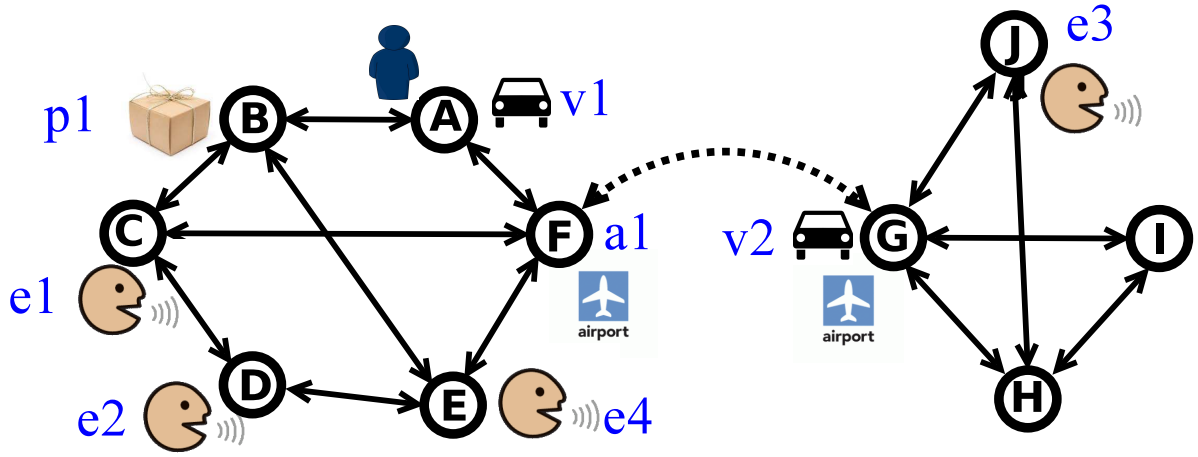

Figure 1: An Example of the Personal Planning Domain

To test on problems with different complexity and properties, four test cases are considered:

1) There are two goals: (at-own **A**) and (at $p1$ **I**). In other words, the agent needs to deliver package $p1$ from **B** to **I**, and then comes back to **A**.

2) There are following goals: (at-own **A**), (is-meet $e1$), (is-meet $e2$), (is-meet $e3$), and (is-meet $e4$). In other words, the agent needs to visit all four experts, and then comes back to **A**.

3) There are following goals: (at-own **A**), (at $p1$ **I**), (is-meet $e1$), (is-meet $e2$), (is-meet $e3$), and (is-meet $e4$). The goal set is a union set of the goals in cases 1) and 2).

4) There are the same goals in case 3). The only difference is two additionally conflicting constraints (conflict $e2$ $e1$), (conflict $e4$ $e3$), which limits some orders for visiting experts.

The FF planner was tested on these cases. It found feasible plans for all four cases:

Case 1) 1. (DRIVE V1 A B) 2. (LOAD V1 P1 B) 3. (DRIVE V1 B E) 4. (DRIVE V1 E F) 5. (UNLOAD V1 P1 F) 6. (LOAD A1 P1 F) 7. (FLY A1 F G) 8. (UNLOAD A1 P1 G) 9. (LOAD V2 P1 G) 10. (DRIVE V2 G I) 11. (UNLOAD V2 P1 I) 12. (DRIVE V2 I G) 13. (FLY A1 G F) 14. (DRIVE V1 F A)

Case 2) 1. (DRIVE V1 A F) 2. (DRIVE V1 F C) 3. (MEET E1 C) 4. (DRIVE V1 C F) 5. (DRIVE V1 F E) 6. (MEET E4 E) 7. (DRIVE V1 E D) 8. (MEET E2 D) 9. (DRIVE V1 D E) 10. (DRIVE V1 E F) 11. (FLY A1 F G) 12. (DRIVE V2 G J) 13. (MEET E3 J) 14. (DRIVE V2 J G) 15. (FLY A1 G F) 16. (DRIVE V1 F A)

Case 3) 1. (DRIVE V1 A B) 2. (LOAD V1 P1 B) 3. (DRIVE V1 B C) 4. (MEET E1 C) 5. (DRIVE V1 C D) 6. (MEET E2 D) 7. (DRIVE V1 D E) 8. (MEET E4 E) 9. (DRIVE V1 E F) 10. (UNLOAD V1 P1 F) 11. (LOAD A1 P1 F) 12. (FLY A1 F G) 13. (UNLOAD A1 P1 G) 14. (LOAD V2 P1 G) 15. (DRIVE V2 G I) 16. (UNLOAD V2 P1 I) 17. (DRIVE V2 I G) 18. (DRIVE V2 G J) 19. (MEET E3 J) 20. (DRIVE V2 J G) 21. (FLY A1 G F) 22. (DRIVE V1 F A)

Case 4) 1. (DRIVE V1 A F) 2. (DRIVE V1 F E) 3. (MEET E4 E) 4. (DRIVE V1 E D) 5. (MEET E2 D) 6. (DRIVE V1 D E) 7. (DRIVE V1 E F) 8. (DRIVE V1 F C) 9. (MEET E1 C) 10. (DRIVE V1 C F) 11. (FLY A1 F G) 12. (DRIVE V2 G J) 13. (MEET E3 J) 14. (DRIVE V2 J G) 15. (FLY A1 G F) 16. (DRIVE V1 F E) 17. (DRIVE V1 E B) 18. (LOAD V1 P1 B) 19. (DRIVE V1 B E) 20. (DRIVE V1 E F) 21. (UNLOAD V1 P1 F) 22. (LOAD A1 P1 F) 23. (FLY A1 F G) 24. (UNLOAD A1 P1 G) 25. (LOAD V2 P1 G) 26. (DRIVE V2 G I) 27. (UNLOAD V2 P1 I) 28. (DRIVE V2 I G) 29. (FLY A1 G F) 30. (DRIVE V1 F A)

Case 1 is a simple extension of the Logistics domain. In the plan, it solves the traditional Logistic problem in Steps 1-11, and then returns to location **A** in Steps 12-14. This plan is optimal.

Case 2 essentially says that the agent must visit all locations where the experts are. Surprisingly, for this simple case, FF could not find the optimal solution, which should be: 1. (DRIVE V1 A B) 2. (DRIVE V1 B C) 3. (MEET E1 C) 4. (DRIVE V1 C D) 5. (MEET E2 D) 6. (DRIVE V1 D E) 7. (MEET E4 E) 8. (DRIVE V1 E F) 9. (FLY A1 F G) 10. (DRIVE V2 G J) 11. (MEET E3 J) 12. (DRIVE V2 J G) 13. (FLY A1 G F) 14. (DRIVE V1 F A).

Case 3 essentially says that the agent must visit all locations where the experts are. For this case, FF could find the optimal solution, possibly because the task of delivering package $p1$ provide a strong bias to arrive location **B** first.

Compared to Case 3, Case 4 applies two additionally conflicting constraints, which essentially say that the agent should meet $e2$ before $e1$, and meet $e4$ before $e3$. For this case, FF could only generate a solution that is far from the optimal solution, which should be: 1. (DRIVE V1 A B) 2. (LOAD V1 P1 B) 3. (DRIVE V1 B E) 4. (MEET E4 E) 5. (DRIVE V1 E D) 6. (MEET E2 D) 7. (DRIVE V1 D C) 8. (MEET E1 C) 9. (DRIVE V1 C F) 10. (UNLOAD V1 P1 F) 11. (LOAD A1 P1 F) 12. (FLY A1 F G) 13. (UNLOAD A1 P1 G) 14. (LOAD V2 P1 G) 15. (DRIVE V2 G I) 16. (UNLOAD V2 P1 I) 17. (DRIVE V2 I G) 18. (DRIVE V2 G J) 19. (MEET E3 J) 20. (DRIVE V2 J G) 21. (FLY A1 G F) 22. (DRIVE V1 F A).

In summary, it has been shown that FF can obtain feasible plans for these cases very quickly. However, if adding goals that require visiting cities in some orders, FF might find the plans far from the optimal. Therefore, it is nontrivial to investigate more competent heuristics for handling these novel situations. In the framework of FF-Replan, probabilistic effects might be handled by

calling FF on a carefully constructed deterministic variant of the planning problem, selecting actions according to the plan, and replanning as well as monitoring an unexpected effect. Next step, the personal planning problem might include more complex features: (1) time resource required for each action is variable; (2) there is constraint on start time and there is deadline for some tasks; (3) beyond minimizing time costs, the agent can be over-subscribed.

References

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

[2] B. Bonet and H. Geffner. Planning as heuristic search. *Artificial Intelligence*, 129(1):5–33, 2001.

[3] T. Bylander. The computational complexity of propositional strips planning. *Artificial Intelligence*, 69(1):165–204, 1994.

[4] S. Edelkamp and M. Helmert. MIPS: The model-checking integrated planning system. *AI Magazine*, 22(3):67, 2001.

[5] K. Erol, D. S. Nau, and V. S. Subrahmanian. Complexity, decidability and undecidability results for domain-independent planning. *Artificial Intelligence*, 76(1):75–88, 1995.

[6] R. E. Fikes and N. J. Nilsson. STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial intelligence*, 2(3):189–208, 1972.

[7] M. Fox and D. Long. PDDL2.1: An extension to pddl for expressing temporal planning domains. *Journal of Artificial Intelligence Research*, 20:61–124, 2003.

[8] M. Helmert. Complexity results for standard benchmark domains in planning. *Artificial Intelligence*, 143(2):219–262, 2003.

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

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

[11] L. P. Kaelbling, M. L. Littman, and A. R. Cassandra. Planning and acting in partially observable stochastic domains. *Artificial intelligence*, 101(1):99–134, 1998.

[12] L. P. Kaelbling, M. L. Littman, and A. W. Moore. Reinforcement learning: A survey. *Journal of Artificial Intelligence Research*, 4:237–285, 1996.

[13] H. Kautz and B. Selman. Pushing the envelope: Planning, propositional logic, and stochastic search. In *National Conference on Artificial Intelligence*, pages 1194–1201, 1996.

[14] H. Kautz and B. Selman. Unifying sat-based and graph-based planning. In *International Joint Conference on Artifical Intelligence*, pages 318–325, 1999.

[15] S. Richter and M. Westphal. The LAMA planner: Guiding cost-based anytime planning with landmarks. *Journal of Artificial Intelligence Research*, 39(1):127–177, 2010.

[16] T. Smith and R. Simmons. Heuristic search value iteration for POMDPs. In *Conference on Uncertainty in Artificial Intelligence*, pages 520–527, 2004.

[17] T. Smith and R. Simmons. Focused real-time dynamic programming for MDPs: Squeezing more out of a heuristic. In *National Conference on Artificial Intelligence*, pages 1227–1232, 2006.

[18] M. M. Veloso. *Learning by Analogical Reasoning in General Problem Solving*. PhD thesis, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA, 1992.

[19] M. M. Veloso, J. Carbonell, A. Perez, D. Borrajo, E. Fink, and J. Blythe. Integrating planning and learning: The prodigy architecture. *Journal of Experimental & Theoretical Artificial Intelligence*, 7(1):81–120, 1995.

[20] D. S. Weld. An introduction to least commitment planning. *AI Magazine*, 15(4):27, 1994.

[21] S. W. Yoon, A. Fern, and R. Givan. Ff-replan: A baseline for probabilistic planning. In *International Conference on Automated Planning and Scheduling*, pages 352–359, 2007.