Plan and goal structure reconstruction: an automated and incremental method based on observation of a single agent

Bartłomiej Józef Dzieńkowski, Urszula Markowska-Kaczmar

Wroclaw University of Technology, Poland
{bartlomiej.dzienkowski,urszula.markowska-kaczmar}@pwr.wroc.pl

Abstract. Plan reconstruction is a task that appears in plan recognition and learning by practice. The document introduces a method of building a goal graph, which holds a reconstructed plan structure, based on analysis of data provided by observation of an agent. The approach is STRIPS-free and uses only a little knowledge about an environment. The process is automatic, thus, it does not require manual annotation of observations. The paper provides details of the developed algorithm. In the experimental study properties of a goal graph were evaluated. Possible application areas of the method are described.

Key words: data analysis, hierarchical plan, sub-goals, reconstruction, recognition, machine learning, agent system

1 Introduction

Building a plan in complex environments is a difficult task. Usually a system designer simplifies a world model, thus, it can be rigidly constrained by a designer's knowledge. In practice this approach gives acceptable results. The process can be improved by analysis and use of existing expert plans. There are sources from which we can mine new plans from the perspective of an observer. However, we do not have an access to some internal parts of the model, especially to a communication channel between planning and execution modules. In fact, we can analyze events in the environment to guess sub-goals of observed agents. Here the task is similar to plan recognition. However, often plan recognition is limited to predefined goals and tasks [3]. Thus, we would like to provide a method that automatically extracts reusable plan elements from a set of recorded observations. We believe that avoiding predefined STRIPS-like conditions and operators increases model flexibility [11].

The following section of the paper motivates the undertaken subject and indicates possible fields of application. Next, references to other works and the state of art is briefly introduced. The next part of the document provides details of the developed method. Subsequently, study results are presented. The final sections are conclusions and further work.

2 Motivation

The primary objective of the study is to provide a method that supports planning by utilizing existing data containing effective solution templates. Moreover, the crucial assumption is to deliver a solution which requires a minimal amount of predefined knowledge, so that the model can become more flexible and domain independent.

In order go give a better understanding of the target application domain, we refer to a general example. Let's assume there is an abstract environment in which agents continuously fulfill their goals. The agents are black-boxes characterized by a high level of intelligence. We are interested in their work and solutions they generate because we can use them for solving our own problems related to the same or an analogical environment. However, we do not have enough resources to build a complete agent model since it is too complex.

The agent's actions are the result of its decisions. Subsequently, the decisions are the product of a plan. A nontrivial task is to reconstruct a plan on the basis of observed actions and events that occur in an environment. Furthermore, a structure of sub-goals inside a plan can be updated and extended when new observations are provided.

The developed method collects plans from observed expert agents and merges them into a map of goals connected by observed solutions. It supports transferring knowledge from experts and learning by practice approaches. However, the algorithm can be applied in many fields. It can be used in network infrastructures for predicting an attacker's goals and sub-goals based on historical intrusions [1][2]. Another example is building a rescue robot or other a human supporting machine by utilizing recorded human experts' actions [5][4][6].

3 State of the art

Mining data from an expert's actions have been previously done in several papers [7][9]. However, those approaches are based on STRIPS-like operators. In this case the method is based on a general environment state representation, which is an alternative approach. In fact, STRIPS operators (or other planning concept) can be added later, but for the purpose of recorded data analysis our algorithm focuses on an encoded sequence of environment states.

Defining a set of STRIPS operators is an additional task to accomplish during development of a system and it can be difficult for complex environment models. Not only operators, but also preconditions cause problems. In paper [7] authors reported a problem with overgrowth of negative preconditions that detracted quality of solutions. Finally, every predefined data greatly constrains system flexibility. In fact, STRIPS provides human-readable planning rules, but it is not perfectly suitable for automatically extracted state transition relations and state transition operators.

Comparing to [7] our approach does not require a set of expert annotations for each observation, or any other effort related to manual marking of recorded data. All the provided input data is processed automatically.

4 Method description

This section provides details and a pseudo-code of the developed method. However, before we proceed to the detailed description of the algorithm a general overview on a model is required.

4.1 General overview

The model consists of an environment and modules that share data flow (Fig. 1). The environment is a well-defined part of an agent system exposed to the observer module, which tracks and records events. The module holds a repository of recorded state sequences. Our algorithm is located in the plan reconstruction module. The output of the module is passed to the planner. An agent controlled by the planner can be placed in the same or a similar environment.

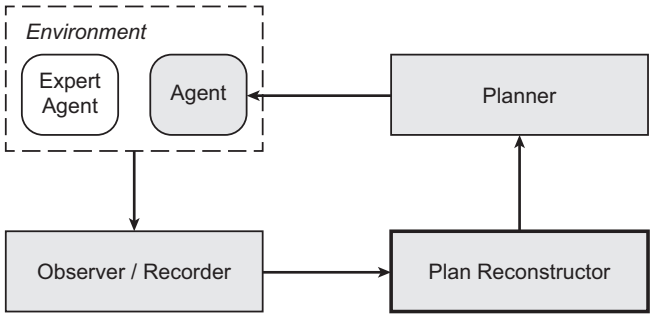

Fig. 1. Module organization and data flow of the discussed model.

The method accepts a set of recorded state sequences on input and returns a goal graph on output. Each input state sequence is an ordered list of encoded environment states collected by a recorder component during a single simulation run. A sequence covers the environment state from the initial to the final state in which agent accomplishes its final task.

A returned goal graph represents a structure of directed connections between environment goal states. It enables us to find the shortest path from the initial to the final state. At the same time it provides a hierarchy of goals and information about method of their accomplishment. In fact, two types of nodes can be found. The first one is a Key Node that stands for a goal state. The second, a Transitive Node is used for connecting Key Nodes. A sequence of adjacent Transitive Nodes between two Key Nodes is a path that stores simple state-change information. Subsequently – a goal graph is a data that can be successfully applied in hierarchical planning methods [10]. However, in this paper we focus on the description of building a goal graph on the basis of an input set of state sequences.

state	–	encoded state of the environment
StateSequence	–	sequence of encoded states
SetOfStateSequences	–	set of state sequences
KEY	–	label for a Key State
TRANSITIVE	–	label for a Transitive State
GoalGraph	–	directed graph in which each node stores a value and a list of references to child nodes

Table 1. Symbol description.

4.2 Algorithm details

The algorithm consists of several steps. Its general framework is presented by Alg. 1 (look for the symbol description in Table 1). The first step is to initialize an empty graph data structure. Next, each state sequence is preprocessed by marking all elements as a Key or Transitive State (in analogy to a Key and Transitive Node). If the goal graph is still empty, the first element from the first state sequence is inserted as the root node. In the same loop the state sequence is broken into a set of paths each of which is a sub-sequence that starts and ends with a Key State – the algorithm is straightforward and specified by Alg. 3. Each sub-sequence (path) is attached to the goal graph (Fig. 2).

Alg. 1 Build Goal Graph from a Set of State Sequences

function buildGoalGraph(*SetOfStateSequences*)
1: *GoalGraph* := ⟨*value* = **nil**, *children* = {}⟩
2: **for all** *StateSequence* **in** *SetOfStateSequences* **do**
3: *StateSequence* := markStates(*StateSequence*)
4: **if** *GoalGraph.value* = **nil then**
5: *GoalGraph.value* := *StateSequence*.getFirstElem()
6: **end if**
7: *Paths* := breakApart(*StateSequence*)
8: **for all** *path* **in** *Paths* **do**
9: *GoalGraph* := attachPath(*GoalGraph*, *path*)
10: **end for**
11: **end for**
12: **return** *GoalGraph*

The method of marking states as a Key or Transitive State showed by Alg. 2 depends on environment characteristics. In general, when we consider some dynamic environment, we should focus on environment state properties that an agent has modified. We can assume that these properties are more important, because they are somehow related to tasks of an agent. Thus, a modification applied to the environment state can be recognized as fulfillment of a goal. A specific approach for marking states problem is described in the next section.

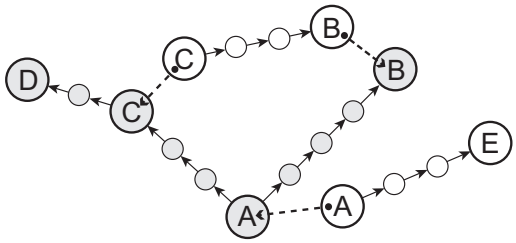

Fig. 2. An example of a goal graph. The larger circles are Key Nodes and the smaller are Transitive Nodes. Dashed arrows show places where new paths are attached to the goal graph. In this case A is the initial node, $\{B, D, E\}$ are the final nodes and C can be considered as a sub-goal.

Alg. 2 Mark States

function markStates(*StateSequence*)

for all *state* **in** *StateSequence* **do**
  find state properties modified by agent and mark them as important, ignore the rest
  if modification has permanent or long term effect
   $\vee$ *state* = *StateSequence*.getFirstElem() **then**
    mark *state* as KEY
  else
    mark *state* as TRANSITIVE
  end if
end for
return *StateSequence*

Alg. 3 Break Apart State Sequence

function breakApart(*StateSequence*)

Paths := {}
path := ⟨⟩
for all *state* **in** *StateSequence* **do**
  path.add(*state*)
  if *state* ≠ *StateSequence*.getFirstElem() **then**
    if *state* **is** KEY **then**
      Paths.add(*path*)
      path := ⟨⟩
      path.add(*state*)
    end if
  end if
end for
return *Paths*

The last part of the method, attaching a path to a goal graph is presented by Alg. 4. Since a node in a goal graph holds a state representation, we require a special method of node comparison that provides some level of generality. In other words, we are more interested in checking whether two nodes are equivalent and represent a very similar situation in the environment rather than finding out whether they are identical. This can be done referring to the previously mentioned state property importance concept. The comparison method is used for searching nodes in a goal graph. It is used for the first time for collecting all nodes equivalent to the first element of the input path, we call them a set of start nodes. In the next step, we use the same method of finding a set of end nodes each of which is equivalent to the last element of the input path. These two node sets enables us to connect gaps between each of two start and end nodes using the input path. In case a connection between two nodes already exists, it is replaced by a shorter path. If the set of end nodes is empty, the input path is simply attached to the start node.

Alg. 4 Attach Path to Goal Graph

```
function attachPath(GoalGraph, path)
 1: // value comparison is made according to the importance of state properties
 2: StartNodes := GoalGraph.findNodesWithValue(path.getFirstElem())
 3: EndNodes := GoalGraph.findNodesWithValue(path.getLastElem())
 4: for all startNode in StartNodes do
 5:    for all endNode in EndNodes do
 6:       GoalGraph.connectNodes(startNode, endNode, path)
 7:    end for
 8:    if EndNodes.getCount() = 0 then
 9:       GoalGraph.attachPath(startNode, path)
10:    end if
11: end for
12: return GoalGraph
```

The described algorithm is a set of general steps that can be adjusted according to some special cases. However, there is an additional feature of the method that is worth mentioning. The algorithm was showed to process all of the state sequences in a single block. It is possible to organize the algorithm flow to work incrementally. Thus, new state sequences can be attached to a goal graph online. A set of sequences will produce the same result no matter the order of attaching to a graph. Additionally, because duplicate paths in a goal graph can be replaced, the graph is not expanding infinitely.

5 Test environment

For the purpose of the research we have developed an agent environment, which is described in this section. The system was designed to cover a general case

of a possible application, but maintaining its simplicity. The environment space is a discrete grid. Each space cell is a resource, but some of them are blocked or their acquisition causes a special effect in the environment. In fact, there are five kinds of resources: an empty space, a permanently blocked space, a gate (a temporarily blocked space), a trigger (locks and unlocks gate), and an objective. A configuration of the environment used in the experiment is presented in Fig. 3. An agent starts a simulation run always in the same position cell. It is able to acquire and move to an unblocked space resource provided that it is in an adjacent cell. If an agent enters a trigger resource, the trigger's internal state $\in \{0, 1\}$ is switched. Simultaneously, a gate linked to the trigger changes its blocking state. A simulation run reaches the end when an agent acquires the objective resource.

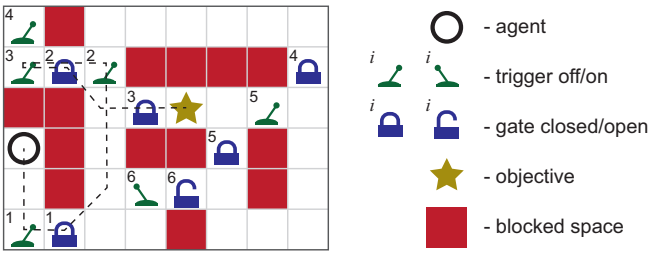

Fig. 3. A resource distribution of an example environment. The dashed line follows an agent's path from the initial position to the objective in 13 steps. A trigger opens (or closes) a gate with the same index number.

An environment state is encoded as a two-dimensional array of state properties. Each state property in the array corresponds to a cell in the space grid. A state property stores a type of a resource and its internal state. Additionally, it shows a position of an agent. Despite the fact that the resource acquisition action can be invoked at any time and it has assigned an execution time, the observer module records only state changes.

Regarding the state marking method from the previous section, in this specific environment it is reasonable to mark state as a Key State whenever an agent uses a trigger or reaches an objective. However, there is still an issue of comparing equivalent states. The problem can be solved by assigning an importance flag to a state property. When an agent modifies a state property for the first time, the property is set to important. Fig. 4 explains the procedure by an example. Hereby, each important state property must match between two states to be considered as equivalent (Fig. 5). The comparison function also checks a stored position of an anent to maintain consistency of a goal graph.

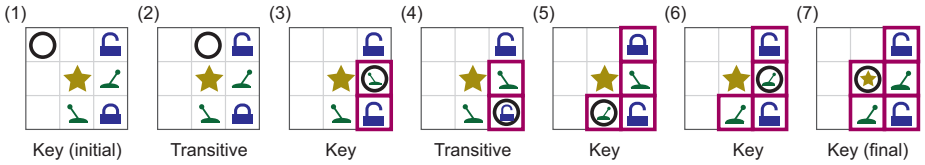

Fig. 4. An example sequence of 7 states. Each state is labelled as a Key or Transitive State. Cells surrounded with a bold border are state properties marked as important.

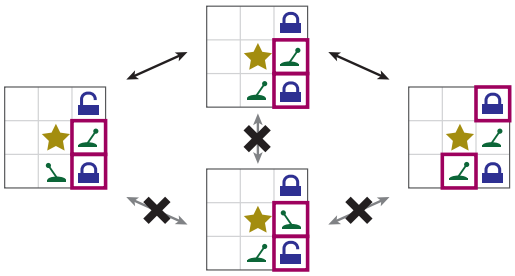

Fig. 5. Equivalency between two states (indicated by an arrow). States are equivalent when all important properties (cells with bold border) of both states match.

6 Experimental study

This section describes an adopted testing methodology of the algorithm and study results. The goal of the experiment was to study characteristics of output graphs provided by the method. The algorithm was tested on sets of state sequences generated by a randomly acting agent in the environment (Fig. 3). Statistics of an example set of sequences generated using this method are located in Table 2. Instead of preparing a large number of state sequences and reading subsets, the sequences were generated on the fly. In the experiment a different numbers of sequences were provided to the algorithm input. Parameters of output graphs were measured. Tests were repeated 10 times and values averaged.

size	avg. len.	stddev	min. len.	max. len.
2000	316.81	236.53	21	1648

Table 2. Statistics of an example generated set of sequences. The numbers stand for: a size of a set, an average sequence length, standard deviation, the minimum sequence length, and the maximum sequence length.

The first part of the experiment was prepared to measure a size of a goal graph with respect to a number of input sequences. Increase of number of nodes

is visualized in Fig. 6. A number of Transitive Nodes initially grows quickly, but then the increase slows down. Theoretically it is possible the number decreases in the future while shorter paths between Key Nodes are provided. However, in an early phase new connections are still revealed and added. The experiment showed that even for our simple environment a goal graph can look very complex – selected graphs are presented in Fig. 7.

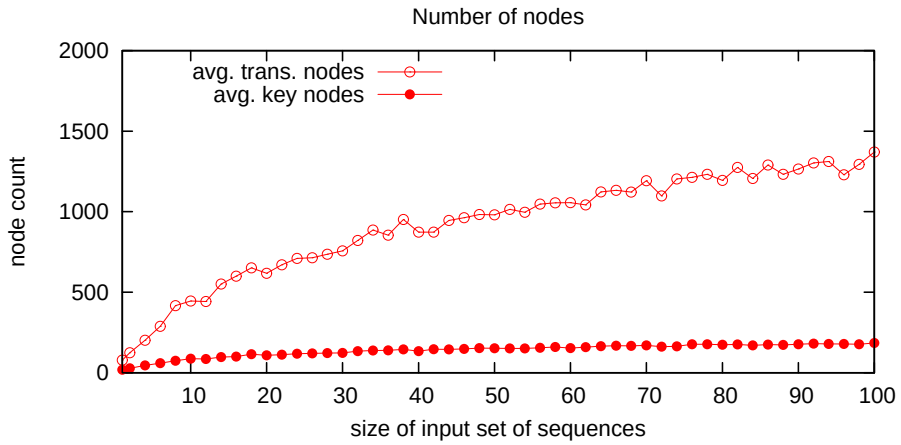

Fig. 6. An average number of transitive and key nodes with respect to a size of an input set of sequences.

Anyhow, Transitive Nodes are insignificant. In fact, they can be pruned since a transition between two connected Key Nodes can be easily found with the aid of a heuristic. More importantly the number of Key Nodes is stable. Even though a number of Key States is proportional to a number of permutation of switchable state properties, complexity of a graph depends on actions of an observed agent.

In the next part of the experiment we have studied how the number of input sequences and graph expansion affects the length of the shortest path between the initial and the final node. The results are visualized in Fig. 8. Clearly an increase of input sequences positively influences a solution quality. Finally, a long-run test showed that a much larger size of input gives no real improvement of the shortest path length (Table 3).

7 Conclusions

The described algorithm differs from other STRIPS-based methods. It requires only basic information about types of elements in an environment. Expert knowledge about relations between objects and their use is compiled into a goal graph. Thus, workload of a system designer is reduced.

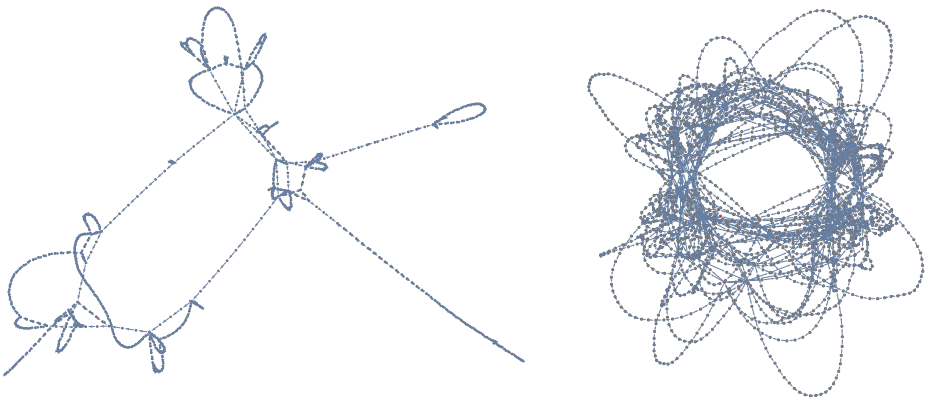

Fig. 7. On the left a goal graph with 546 nodes built from 10 state sequences and on the right a goal graph with 2953 nodes built from 7000 state sequences.

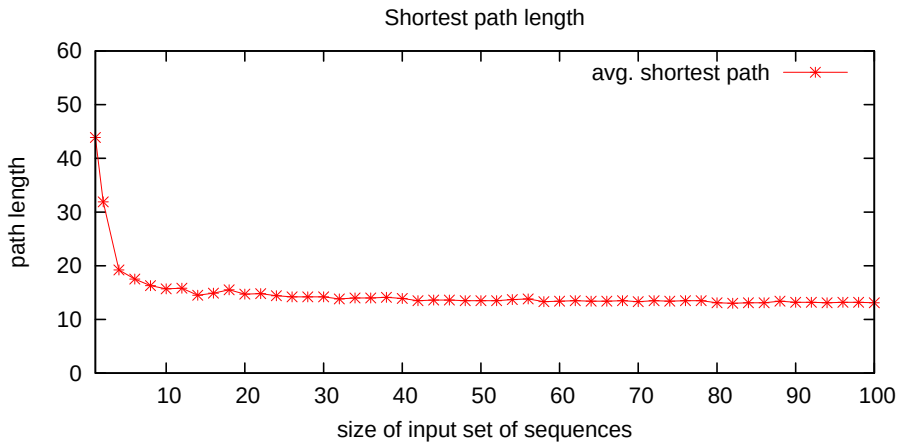

Fig. 8. An average shortest path from the initial to the nearest final node with respect to a size of an input set of sequences.

input set size	avg. key nodes	avg. trans. nodes	avg. shortest path
7000	213	2737.3	13

Table 3. Statistics of the biggest tested input sequences set. The numbers stands for: size of an input set of sequences, average number of key nodes, average number of transitive nodes and shortest path from the initial to the nearest final node.

Despite the fact that the method in some cases can have a limited application, its main advantage is flexibility and automatic data acquisition support. A planner based on the approach is not required to manipulate STRIPS conditions and operators. In fact, it can use a set of general transition functions to move between graph nodes. The set of transitions can be automatically extended based on observed expert solutions.

However, the method is not free from flaws. It requires a large amount of input data. Thus, it can be applied for analysis of existing models which continuously generate data (e.g., multiplayer game matches). Additionally, the algorithm in its original form can lead to performance problems. It is required to utilize some graph search optimizations before applying in practice. Finally, the approach can evolve to a hybrid model.

8 Further work

The described algorithm is dedicated to environments with a single agent, or many agents that work independently. This assumption limits practical application. However, the paper refers to an early stage of the research that aims on an analogical approach but dedicated to a group of cooperating agents. In the new case, the algorithm can no more entirely rely on environment state comparison. A new approach based on cooperation patterns should be proposed. Hereby, not a state change event but an execution of cooperation pattern will link nodes in the goal graph [8]. Finally, in order to provide system flexibility, the cooperation patterns should be automatically recognized and mined.

References

1. Gu, W., Ren, H., Li, B., Liu, Y., Liu, S.: Adversarial Plan Recognition and Opposition Based Tactical Plan Recognition. In: International Conference on Machine Learning and Cybernetics, pp. 499–504. (2006)
2. Gu, W., Yin, J.: The Recognition and Opposition to Multiagent Adversarial Planning. In: International Conference on Machine Learning and Cybernetics, pp. 2759–2764. (2006)
3. Camilleri, G.: A generic formal plan recognition theory. In: International Conference on Information Intelligence and Systems, pp. 540–547. (1999)
4. Takahashi, T., Takeuchi, I., Matsuno, F., Tadokoro, S.: Rescue simulation project and comprehensive disaster simulator architecture. In: International Conference on Intelligent Robots and Systems, vol. 3, pp. 1894–1899. (2000)
5. Takahashi, T., Tadokoro, S.: Working with robots in disasters. In: Robotics and Automation Magazine, IEEE, vol. 9, no. 3, pp. 34–39. (2002)
6. Chernova, S., Orkin, J., Breazel, C.: Crowdsourcing HRI through Online Multiplayer Games. In: Proceedings of AAAI Fall Symposium on Dialog with Robots, pp. 14–19. (2010)
7. Wang, X.: Learning by Observation and Practice: An Incremental Approach for Planning Operator Acquisition. In: Proceedings of the 12th International Conference on Machine Learning, pp. 549–557. (1995)
8. Ehsaei, M., Heydarzadeh, Y., Aslani, S., Haghighat, A.: Pattern-Based Planning System (PBPS): A novel approach for uncertain dynamic multi-agent environments. In: 3rd International Symposium on Wireless Pervasive Computing, pp. 524–528. (2008)
9. Nejati, N., Langley, P., Konik, T.: Learning Hierarchical Task Networks by Observation. In: Proceedings of the 23rd International Conference on Machine Learning, pp. 665–672. (2006)
10. Sacerdoti, Earl D.: A Structure For Plans and Behavior. In: Artificial Intelligence Center, Elsevier North-Holland, Technical Note 109. (1977)
11. Fikes, R., Nilsson, N.: STRIPS: a new approach to the application of theorem proving to problem solving. In: IJCAI, pp. 608–620. (1971)