Flexible Plan-Subplan Matching for Plan Recognition in Case-Based Agents

Keith A. Frazer[1,3], Swaroop S. Vattam[2], and David W. Aha[3]

[1]College of Computing, Georgia Institute of Technology, Atlanta, GA
[2]NRC Postdoctoral Fellow; Naval Research Laboratory (Code 5514); Washington, DC
[3]Navy Center for Applied Research in Artificial Intelligence;
Naval Research Laboratory (Code 5514); Washington, DC
kfrazer3@gatech.edu | {swaroop.vattam.ctr.in, david.aha}@nrl.navy.mil

Abstract. Plan-subplan matching is an important step in case-based plan recognition. We present RelaxedVF2, an algorithm for plan-subplan matching for plans encoded using the Action Sequence Graph representation. RelaxedVF2 is a subgraph monomorphism algorithm that affords flexibility and error tolerance for plan-subplan matching. We present a study comparing RelaxedVF2 with an alternate degree-sequence matcher that we used in our prior work and found that RelaxedVF2 attains higher plan recognition accuracy on a paradigmatic domain.

Keywords: Plan Recognition, Case-Based Reasoning, Action-Sequence Graph, Relaxed Graph Matching, Error-Tolerant

1. Introduction

An agent on a team must cooperate and coordinate its actions with its teammates, requiring the ability to recognize its teammates' plans. *Plan recognition* refers to the task of observing a teammate's current actions, inferring the plan governing those actions, and predicting that teammate's future actions. A plan recognizer takes as input the observed portion of a plan (subplan) and outputs a (predicted) full plan. A case-based plan recognizer matches its input subplan to a set of plans in its case base and retrieves a most similar plan to the given subplan. We assume that the most similar plan best explains the observed subplan. Plan-subplan matching is therefore a key component of case-based plan recognition (CBPR).

The algorithm used for plan-subplan matching depends on the representation of plans. Typically plans are represented as (a sequence of) propositions in first-order predicate logic. We instead use an *Action Sequence Graph (ASG)* representation (Vattam et al., 2014; 2015), which has some nice properties: (1) it captures the topology of the propositional plans using graphs, (2) better lends itself to vectorization and approximate matching, (3) and makes the matching process more robust to input errors (Vattam et al., 2015). ASG represents a plan as a labeled directed multigraph. Plan-subplan matching using ASGs reduces to graph-subgraph matching.

We introduce the RelaxedVF2 algorithm for graph-subgraph matching that is tailored to matching plans represented as ASGs. RelaxedVF2 is an extension of the popular VF2 algorithm (Cordella et al., 2004) for subgraph isomorphism. The extensions to VF2 we propose transform it into a subgraph monomorphism matching algorithm, which makes RelaxedVF2 better suited for additional edges that arise between the nodes of states and actions as a plan's observed execution progresses.

In §2 we present related work on CBPR and graph matching techniques. In §3 we present the ASG representation for plans. In §4 we present our plan-subplan matching approach, including the RelaxedVF2 algorithm and the scoring function. In §5 we present an initial empirical study comparing the performance of RelaxedVF2 to an alternative degree-sequence matching algorithm that we used in our prior work (Vattam et al., 2015). Our results show that RelaxedVF2 compares favorably to the alternative approach. We conclude and discuss future research plans in §6.

2. Related Research

Several approaches has been proposed to address the problem of plan recognition (Sukthankar et al., 2014), including consistency-based (e.g., Hong, 2001; Kautz & Allen, 1986; Kumaran, 2007; Lau et al., 2003; Lesh & Etzioni, 1996), and probabilistic approaches (e.g., Bui, 2003; Charniak & Goldman, 1991; 1993; Geib & Goldman, 2009; Goldman et al., 1999; Pynadath & Wellman, 2000). Both types are "model-heavy", requiring accurate models of an actor's possible actions and how they interact to accomplish different goals. Engineering these models is difficult and time consuming. Furthermore, these plan recognizers perform poorly when confronted with novel situations and are brittle when the operating conditions deviate from model parameters.

CBPR is a model-lite, less studied approach to plan recognition. Existing CBPR approaches (e.g., Cox & Kerkez, 2006; Tecuci & Porter, 2009) eschew generalized models for plan libraries that contain plan instances which can be gathered from experience. CBPR algorithms can respond to novel inputs outside the scope of their plan library by using plan adaptation techniques. However, earlier CBPR approaches were not error-tolerant.

In contrast, our work on SET-PR focuses on error-tolerant CBPR (Vattam et al., 2014; 2015). We showed that SET-PR is robust to three kinds of inputs errors (missing, mislabeled, and extraneous actions). One of the factors contributing to its robustness is that SET-PR uses an ASG plan representation and the degree sequence similarity function for plan-subplan matching. Although we previously showed that SET-PR was robust to input errors, there is room for improvement.

VF2 (Cordella et al., 2004) is an exact graph matching algorithm for finding node-induced subgraph isomorphisms. It is one of the few such algorithms applicable to directed multigraphs. Our extension, RelaxedVF2, transforms VF2 from finding node-induced subgraph isomorphisms to subgraph monomorphisms (see Figure 1 for an illustration on how these differ) and by modifying it to return partial mappings from graph to subgraph when no complete match is available.

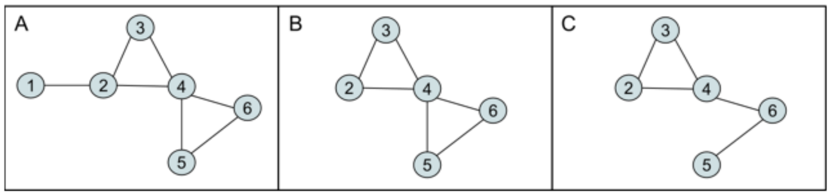

Figure 1: Graph B is a *node-induced isomorphism* of Graph A because it is missing a node (1) but preserves all edges between nodes shared in both Graphs A and B. Graph C is a *monomorphism* of A because it is missing a node (1) and an edge (between 4 and 5). Definitions are provided in Section 4.1.

3. Plan Representation: Action Sequence Graphs

Suppose a plan is modeled as an *action state sequence* $\mathbb{s} = \langle(\boldsymbol{a_0}, \boldsymbol{s_0}), \ldots, (\boldsymbol{a_n}, \boldsymbol{s_n})\rangle$, where each action $\boldsymbol{a_i}$ is a ground operator in the planning domain, and $\boldsymbol{s_i}$ is a ground state obtained by executing $\boldsymbol{a_i}$ in $\boldsymbol{s_{i-1}}$, with the caveat that $\boldsymbol{s_0}$ is an initial state, $\boldsymbol{a_0}$ is null, and $\boldsymbol{s_n}$ is a goal state. An action $\boldsymbol{a}$ in $(\boldsymbol{a}, \boldsymbol{s}) \in \mathbb{s}$ is a ground literal $\boldsymbol{p} = p(o_1{:}\, t_1, \ldots, o_n{:}\, t_n)$, where $p \in \boldsymbol{P}$ (a finite set of predicate symbols), $o_i \in \boldsymbol{O}$ (a finite set of object types), and t_i is an instance of o_i (e.g., stack(block:A, block:B)). A state $\boldsymbol{s}$ in $(\boldsymbol{a}, \boldsymbol{s}) \in \mathbb{s}$ is a set of ground literals (e.g., {on(block:A,block:B), on(block:B,substrate:TABLE)}).

An *Action Sequence Graph (ASG)* is a graphical representation of a plan that preserves its topology (including the order of the propositions and their arguments). Vattam et al. (2014; 2015) provide a detailed definition of ASGs and their generation. An ASG is automatically generated by transforming individual propositions in a plan into predicate encoding graphs, and by taking the union of all the individual predicate encoding graphs so as to maintain the total order of the plan. Figure 2 shows an example proposition and its corresponding predicate encoding graph. Figure 3 shows an example full plan and its corresponding ASG. An ASG is a labeled directed multigraph, which constrains the set of graph matching algorithms that can manipulate them.

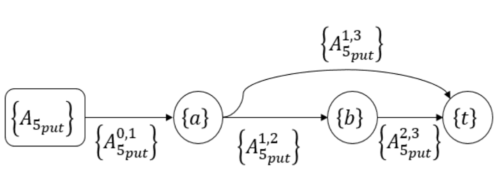

Figure 2: A predicate encoding graph corresponding to $\boldsymbol{p} = put(block{:}\, a, block{:}\, b, table{:}\, t)$

4. Robust Plan-Subplan Matching

As our goal is error-tolerant plan recognition, our approach requires plan-subplan matching that is robust to input errors. Plan-subplan matching requires a measure of similarity. As we encode our plans in the case library and the input subplans as graphs, we utilize maximum common subgraph *monomorphism* as a measure of similarity between them rather than the more conventional maximum common subgraph *isomorphism* measure.

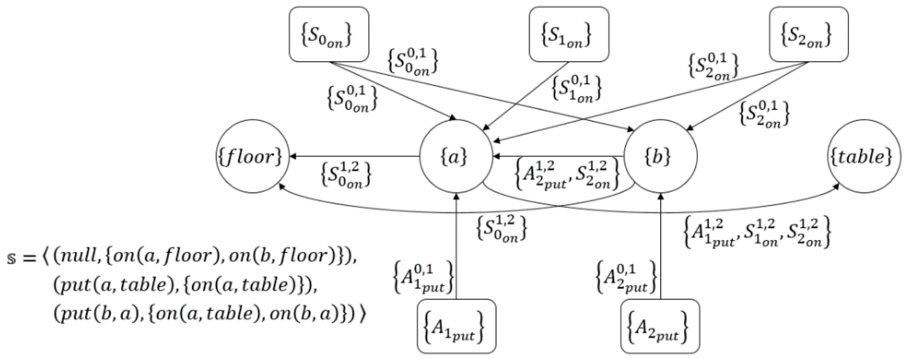

Figure 3: An example of a plan with three action-state sequences and its corresponding ASG

Let G_1, G_2 be graphs composed of sets of vertices and edges V_1, V_2 and E_1, E_2 respectively. G_2 is *isomorphic* to a subgraph of G_1 if there exists a one-to-one mapping between each vertex of V_2 and a vertex in V_1 and the number of edges between nodes in the mapping are maintained. G_2 is instead *monomorphic* if it consists of any subset of the vertices and edges of G_1. Monomorphism must be utilized over isomorphism when matching incomplete subplans to complete plans in the case library because as plans are observed new edges are often added relating existing action and state vertices.

RelaxedVF2 (§4.1), an exact graph matching algorithm, does not return a similarity score. It instead returns a one-to-one mapping of nodes between the subplan and plans in the case library. While the length of the maximum common subgraph is often used to score matches, we instead developed a more nuanced candidate scoring algorithm (§4.2) to increase matching accuracy.

4.1 RelaxedVF2

RelaxedVF2 (Algorithm 1) computes the maximum common subgraph monomorphism between two labeled directed multigraphs. Here we refer to node-induced isomorphism as a subset of the nodes with all corresponding edges between them.

VF2 matches two graphs, G_1 and G_2, using semantic and syntactic feasibility functions to iteratively add compatible nodes of the graphs to an internal mapping, M, which is expressed as a set of pairs (n, m) that represent the mapping of a node $n \in G_1$ with a node $m \in G_2$. Therefore, a mapping M is a *graph* isomorphism if it is a bijective function that preserves the branching structure of the two graphs, and M is a *subgraph* isomorphism if the mapping is an isomorphism between G_2 and a subgraph of G_1.

The original VF2 algorithm uses five syntactic feasibility rules to check if a pair (n, m) can be included in M. These rules are listed in Table 1.

Table 1: Syntactic feasibility rules for VF2 and RelaxedVF2

VF2	**RelaxedVF2**
$R_{pred}(s,n,m) \Leftrightarrow$ $\big(\forall n' \in M_1(s) \cap Pred(G_1,n) \exists m' \in Pred(G_2,m) \mid (n',m') \in M(s)\big) \wedge \big(\forall n' \in M_2(s) \cap Pred(G_2,n) \exists m' \in Pred(G_1,m) \mid (n',m') \in M(s)\big)$	$R_{pred}(s,n,m) \Leftrightarrow$ $\left(\begin{matrix}\forall n' \in M_2(s) \cap Pred(G_2,n) \\ \exists m' \in Pred(G_1,m) \mid (n',m') \in M(s)\end{matrix}\right)$
$R_{succ}(s,n,m) \Leftrightarrow$ $\big(\forall n' \in M_1(s) \cap Succ(G_1,n) \exists m' \in Succ(G_2,m) \mid (n',m') \in M(s)\big) \wedge \big(\forall n' \in M_2(s) \cap Succ(G_2,n) \exists m' \in Succ(G_1,m) \mid (n',m') \in M(s)\big)$	$R_{succ}(s,n,m) \Leftrightarrow$ $\left(\begin{matrix}\forall n' \in M_2(s) \cap Succ(G_2,n) \\ \exists m' \in Succ(G_1,m) \mid (n',m') \in M(s)\end{matrix}\right)$
$R_{in}(s,n,m) \Leftrightarrow$ $(Card(Succ(G\_1,n) \cap T\_1\text{^}in\ (s)) \geq Card(Succ(G\_2,n) \cap T\_2\text{^}in\ (s))) \wedge (Card(Pred(G\_1,n) \cap T\_1\text{^}in\ (s)) \geq Card(Pred(G\_2,n) \cap T\_2\text{^}in\ (s)))$	$R_{in}(s,n,m) \Leftrightarrow$ $(Card(Succ(G\_1,n) \cap T\_1\text{^}in\ (s)) \geq Card(Succ(G\_2,n) \cap T\_2\text{^}in\ (s))) \wedge (Card(Pred(G\_1,n) \cap T\_1\text{^}in\ (s)) \geq Card(Pred(G\_2,n) \cap T\_2\text{^}in\ (s)))$
$R_{out}(s,n,m) \Leftrightarrow$ $(Card(Succ(G\_1,n) \cap T\_1\text{^}out\ (s)) \geq Card(Succ(G\_2,n) \cap T\_2\text{^}out\ (s))) \wedge (Card(Pred(G\_1,n) \cap T\_1\text{^}out\ (s)) \geq Card(Pred(G\_2,n) \cap T\_2\text{^}out\ (s)))$	$R_{out}(s,n,m) \Leftrightarrow$ $(Card(Succ(G\_1,n) \cap T\_1\text{^}out\ (s)) \geq Card(Succ(G\_2,n) \cap T\_2\text{^}out\ (s))) \wedge (Card(Pred(G\_1,n) \cap T\_1\text{^}out\ (s)) \geq Card(Pred(G\_2,n) \cap T\_2\text{^}out\ (s)))$
$R_{out}(s,n,m) \Leftrightarrow$ $(Card(\tilde{N}\_1\ (s) \cap Pred(G\_1,n)) \geq Card(\tilde{N}\_2\ (s) \cap Pred(G\_2,n))) \wedge (Card(\tilde{N}\_1\ (s) \cap Succ(G\_1,n)) \geq Card(\tilde{N}\_2\ (s) \cap Succ(G\_2,n)))$	

The first two rules determine match compatibility based on an equivalent number of incoming and outgoing edges per node. The final three rules look ahead to adjacent nodes to prune the search tree. We adapted VF2 to relax its enforcement of graph/subgraph edge counts while maintaining rules disqualifying additional edges in the subgraph not present in the graph. We removed the fifth rule because strict lookahead rules run counter to our goal of increased error tolerance. We also made modifications to enable returning partial matches, removing the rule that matches must be equal in length to the subgraph.

We optimized RelaxedVF2 for graph recognition, and thus primarily rely on the semantic similarity of node labels to restrict our search space. Our simple semantic feasibility function uses an exact string match of the node labels. Any plan recognizer using this algorithm would need to provide as input its own domain-specific semantic similarity measure. RelaxedVF2 uses a depth-first search through all possible nodes that can be added based on semantic and structural similarity. It then progresses to nodes matching on only semantic similarity before finally adding nodes matching using only structural similarity.

Algorithm 1: RelaxedVF2

```
PROCEDURE RelaxedVF2($s, G_1, G_2$)
  INPUT:     An intermediate state $s$ (the initial state $s_0$ has $M(s_0) = \emptyset$) and two graphs
  OUTPUT:   the mappings between $G_1$ and $G_2$
  IF $M(s)$ covers all the nodes of $G_2$ THEN
    OUTPUT $M(s)$   //The function $M$ returns the mappings between nodes of $G_1, G_2$ in state $s$
  ELSE
    L = [ ]    //Sorted list of feasible pairs
    mappingFound = False
    $P(s) \leftarrow$ candidate pairs for inclusion in $M(s)$    //Used candidate pairs function from VF2
    FOREACH $(n, m) \in P(s)$
      IF $F(s, n, m)$ THEN
        L $\leftarrow$ L $\cup$ $(n, m)$
    WHILE NOT mappingFound    //Loop until match is found or no more candidates
      $s' \leftarrow M(s) \cup$ L.pop$(n, m)$    //Get the top feasible pair from list
      mappingFound = RelaxedVF2($s', G_1, G_2$)    //Recursive call
    IF NOT mappingFound    //Output partial match if no match found
      OUTPUT $M(s)$
    Restore data structures
```

4.2 Scoring

The size of the largest common subgraph can be used as a similarity measure (Bergmann, 2002). VF2 is error tolerant and will return matches even if they are of lower quality. Therefore, we designed a metric that is a function of both match size and quality. The match algorithm scores 1 point for every full match based on both semantics and structure (0.7 per semantic match and 0.3 per structural match, based on previous weights used in SET-PR (Vattam et al., 2015)). After retrieving all matches of the subgraph against the case library this score is then used to sort and find the best match.

5. Empirical Study

In this study, we compare plan-subplan matching using two similarity measures on ASGs: (1) RelaxedVF2, and (2) DSQ (degree-sequence matcher) (Vattam et al., 2014; 2015). Our claim is that RelaxedVF2 offers better performance compared to DSQ.

The default plan representation consists of action-state sequences ($\langle(\boldsymbol{a_0}, \boldsymbol{s_0}), \ldots, (\boldsymbol{a_n}, \boldsymbol{s_n})\rangle$). We also evaluated a plan representation consisting of only action sequences ($\langle(\boldsymbol{a_0}), \ldots, (\boldsymbol{a_n})\rangle$) because state information is not always available in all planning domains and it presents a more difficult challenge for DSQ. This yields four conditions: RelaxedVF2$_{\text{ActionStates}}$, DSQ$_{\text{ActionStates}}$, RelaxedVF2$_{\text{ActionsOnly}}$, and DSQ$_{\text{ActionsOnly}}$.

We empirically test whether, for error tolerant CBPR, using RelaxedVF2 outperforms DSQ for both the ActionStates and ActionsOnly conditions. We use plan recognition accuracy as our performance metric, where accuracy is defined as the ratio of queries that resulted in correct plan retrieval to the total number of queries.

5.1 Empirical method

We conducted our experiments in the Blocks World domain, which is simple and allows us to quickly generate a plan library L with desired characteristics. We used SHOP2 (Nau et al., 2003) to generate L's plans as follows. We generated 20 random initial states and paired each with 5 randomly generated goal states to obtain 100 planning problems. Each was given as input to SHOP2 to obtain a plan. We fixed plan length to 20 by discarding any whose length was not 20 and generating a new one (with a different goal state) in its place. This distribution was chosen because it is challenging for plan recognition.

We evaluated plan recognition accuracy using a leave-one-in strategy (Aha & Breslow, 1997). For each compared condition:

1. We randomly selected a plan π in L (π is *not* deleted from L).
2. We introduced a fixed percentage of error into π consisting of a uniform distribution of missing, mislabeled, and extraneous actions and random distortions of states associated with those actions. The error levels that we tested were $\{0\%,10\%,20\%,30\%,40\%,50\%\}$.
3. The error-π plan was then used to incrementally query L to retrieve a plan. For example, if error-π had 20 steps, the evaluator performed 11 queries at the following plan lengths: 0% (initial state only, no actions are observed), 10% (first two actions and states are observed), and so on until 100% (full plan is observed).
4. Each query derived from error-π was used to retrieve the top matching plan π^{sol}. If π was equal to π^{sol}, it was considered a success and a failure otherwise.
5. We repeated steps 1-4 for all 100 plans in L in each of 20 trials.

This yields 1100 queries per error percent level per trial, yielding 132,000 queries (1100 queries × 6 error levels × 20 trials). We computed average accuracy over 20 trials.

5.2 Results and discussion

We computed mean accuracy for each percentError (in [0.0,0.5] with increments of 0.1) and each percentAction (in [0.0,1.0] with increments of 0.1) for RelaxedVF2 and DSQ. The results are shown in Figures 4 and 5 for ActionStates and ActionsOnly, respectively. Our results show that for all error levels and percent actions RelaxedVF2's mean accuracy was higher than DSQ's. In ActionStates, RelaxedVF2 achieves 50% accuracy by 20% actions at all error levels, but DSQ only achieves 50% accuracy at 100% actions at only 0% and 10% error. In ActionsOnly, RelaxedVF2 achieves 50% accuracy by 40% actions at all error levels, but DSQ only approaches 50% accuracy at 100% actions with 0% error.

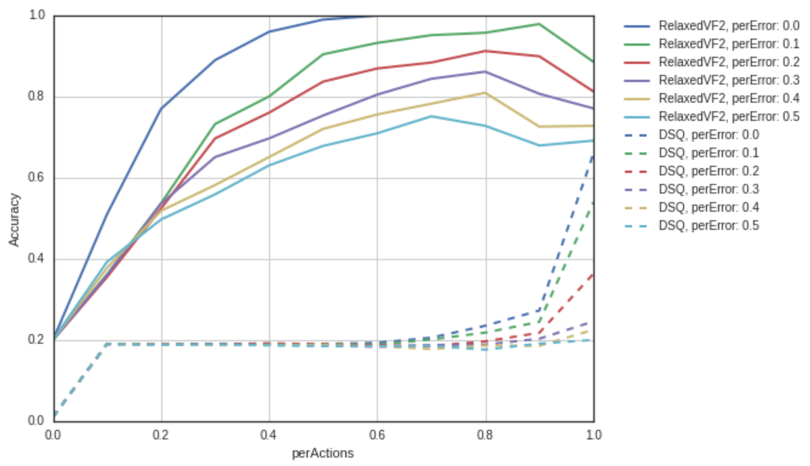

Figure 4: Mean plan recognition accuracy for the ActionStates conditions

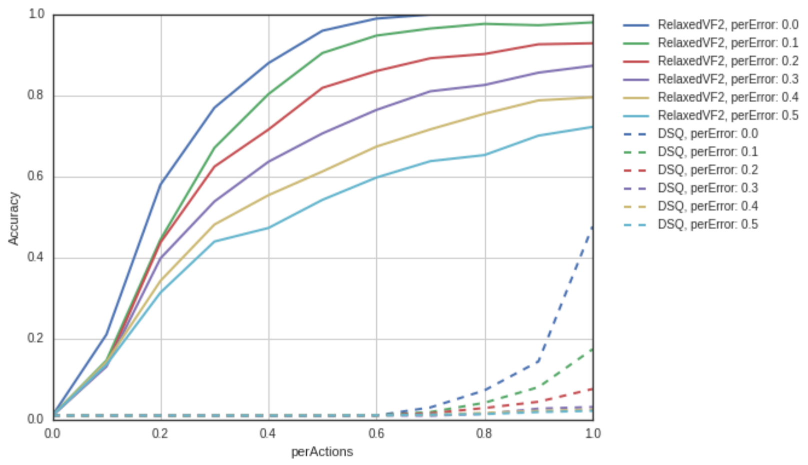

Figure 5: Mean plan recognition accuracy for the ActionsOnly conditions

We conducted a one-way ANOVA test to compare the effects of percent Actions (0%-100%), percent Error (0%-50%) and the matching algorithms (RelaxedVF2, DSQ) on accuracy. There was a significant effect on accuracy at $p < 0.05$ with respect to the matching algorithms (F(1,17)=18687550.204, p=0.0). This analysis shows that RelaxedVF2 significantly outperformed DSQ, which lends support to our claim.

DSQ performs considerably worse without state information because the ASGs become much smaller. The degree sequences across the partitions of the smaller graphs will yield similar values, preventing DSQ from disambiguating the different plans.

Surprisingly, accuracy decreased around 80% actions in RelaxedVF2 in the ActionStates condition (Figure 4). As the error level increases this dip occurs earlier. We hypothesize that the more densely connected graphs resulting from additional action-state information causes these graphs to more closely resemble each other, thus reducing recognition accuracy. We plan to investigate this in future work.

Given that RelaxedVF2 is an exact graph matching algorithm and DSQ is an approximate algorithm, DSQ should have a significantly shorter runtime. In this study, RelaxedVF2 had a mean runtime (in seconds) of 0.121 and 0.045 in the ActionStates and ActionsOnly conditions, respectively. DSQ mean runtime was 0.020 and 0.019 in these conditions. We subjected the mean runtimes to a *t*-test and found the differences in the runtimes to be significant at $p < 0.05$ for both conditions.

6. Summary

CBPR under imperfect observability requires error tolerant plan-subplan matching, which requires flexible representation and matching algorithms. In earlier work we introduced the ASG representation for plan recognition and degree-sequence plan matching (Vattam et al., 2014; 2015). Although this matching algorithm worked reasonably well, there remained room for improvement. Here we presented RelaxedVF2, an alternative plan-subplan matching algorithm. It is a subgraph monomorphism algorithm, and thus affords flexibility and error tolerance in matching compared to VF2. In our empirical study we found support for our claim that, for error-tolerant CBPR, RelaxedVF2 can outperform the degree-sequence matcher, at least for the paradigmatic domain we studied.

In future work, we will investigate whether the same result occurs when using datasets from additional domains to address the single dataset limitation of our current study. We also plan to integrate RelaxedVF2 into our plan recognition architecture to complement the existing methods. We also plan to do a comparative study with other state-of-the-art plan recognizers.

Acknowledgements

Thanks to OSD ASD (R&E) for sponsoring this research. Swaroop Vattam conducted this research while an NRC post-doctoral research associate at NRL. Keith Frazer conducted this research while an NREIP intern at NRL. The views and opinions in this paper are those of the authors and should not be interpreted as representing the official views of NRL or OSD.

References

Aha, D. W., & Breslow, L. A. (1997). Refining conversational case libraries. *Proceedings of the Second International Conference on CBR* (pp. 267-278). Providence, RI: Springer-Verlag.

Bergmann, R. (2002). *Experience management: Foundations, development methodology, and Internet-based applications*. Berlin, Germany: Springer-Verlag.

Bui, H. (2003). A general model for online probabilistic plan recognition. *Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence* (pp. 1309–1315). Acapulco, Mexico: Morgan Kaufmann.

Charniak, E., & Goldman, R. (1991). A probabilistic model of plan recognition. P*roceedings of the Ninth National Conference on Artificial Intelligence* (pp. 160–165). Anaheim, CA: AAAI Press.

Charniak, E., & Goldman, R. (1993). A Bayesian model of plan recognition. *Artificial Intelligence*, *64*, 53–79.

Cordella, L., P., Foggia, P., Sansone, C., & Vento, M. (2004). A (sub)graph isomorphism algorithm for matching large graphs. *Transactions on Pattern Analysis and Machine Intelligence*, *26*(10), 1367-1372.

Cox, M.T., & Kerkez, B. (2006). Case-based plan recognition with novel input. *Control and Intelligent Systems*, *34*(2), 96–104

Geib, C.W., & Goldman, R.P. (2009). A probabilistic plan recognition algorithm based on plan tree grammars. *Artificial Intelligence*, *173*(11), 1101–1132.

Goldman, R.P., Geib, C.W., & Miller, C.A. (1999). A new model of plan recognition. *Proceedings of the Fifteenth Conference on Uncertainty in Artificial Intelligence* (pp. 245–254). Bled, Slovenia: Morgan Kaufmann.

Hong, J. (2001). Goal recognition through goal graph analysis. *Journal of Artificial Intelligence Research*, *15*, 1–30.

Kautz, H., & Allen, J. (1986). Generalized plan recognition. *Proceedings of the Fifth National Conference on Artificial Intelligence* (pp. 32–37). Philadelphia, PA: Morgan Kaufmann.

Kumaran, V. (2007). *Plan recognition as candidate space search*. Master's thesis, Department of Computer Science, North Carolina State University, Raleigh, NC.

Lau, T., Wolfman, S.A., Domingos, P., & Weld, D.S. (2003). Programming by demonstration using version space algebra. *Machine Learning*, *53*(1-2), 111–156.

Lesh, N., & Etzioni, O. (1996). Scaling up goal recognition. *Proceedings of the Fifth International Conference on Knowledge Representation and Reasoning* (pp. 178–189). Cambridge, MA: Morgan Kaufmann.

Nau, D., Au, T.-C., Ilghami, O, Kuter, U, Murdock, J.W., Wu, D., & Yaman, F. (2003). SHOP2: An HTN planning system. *Journal of Artificial Intelligence Research*, *20*, 379-404.

Pynadath, D.V., & Wellman, M.P. (2000). Probabilistic state-dependent grammars for plan recognition. P*roceedings of the Conference on Uncertainty in Artificial Intelligence* (pp. 507–514). San Francisco, CA: Morgan Kaufmann.

Sukthankar, G., Goldman, R., Geib, C., Pynadath, D., & Bui, H. (Eds.) (2014). *Plan, Activity, and Intent Recognition*. Cambridge, MA: Elsevier.

Tecuci, D., & Porter, B.W. (2009). Memory based goal schema recognition. In *Proceedings of the 22nd International Florida AI Research Society Conference*. Sanibel Island, FL: AAAI Press.

Vattam, S.S., Aha, D.W., & Floyd, M. (2014). Case-based plan recognition using action sequence graphs. *Proceedings of the Twenty-Second International Conference on Case-Based Reasoning* (pp. 495-510). Cork, Ireland: Springer.

Vattam, S., Aha, D.W., & Floyd, M. (2015). Error tolerant plan recognition: An empirical investigation. In *Proceedings of the Twenty-Eighth Florida Artificial Intelligence Research Society Conference*. Hollywood, FL: AAAI Press.