Increasing the Runtime Speed of Case-Based Plan Recognition

Michael Maynord
Computer Science Department
University of Maryland
College Park, MD 20742
maynord@umd.edu

Swaroop Vattam[1,2] **& David W. Aha**[2]
[1]NRC Postdoctoral Research Fellow
[2]Navy Center for Applied Research in AI
Naval Research Laboratory, Code 5514
Washington, DC 20375
{swaroop.vattam.ctr.in,david.aha}@nrl.navy.mil

Abstract

We present PPC (Plan Projection and Clustering), an algorithm that creates a plan hierarchy for case-based plan recognition systems. PPC is motivated by a desire to improve the response time of robots working in collaboration with humans. It projects the plans of a case base into a Euclidean space and iteratively clusters plans within that space, producing an abstraction hierarchy. Case retrieval traverses down this hierarchy, and requires fewer comparisons than a search of the corresponding flat case base. Our approach also has the advantage that it does not require substantial domain knowledge. We report PPC's empirical performance on synthetically generated plans, showing that it increases runtime speed without substantially reducing plan retrieval accuracy when the plans are generated using a non-random distribution.

Introduction

We are developing an intelligent agent to control, in simulated scenarios, a robot that teams with a small detachment of soldiers on a reconnaissance mission. The agent communicates with its human teammates, and is expected to respond to situations appropriately when no commands are or can be given. For example, if the detachment comes under enemy fire the robot should respond appropriately (e.g., position itself between the enemy and its team) and autonomously.

To detect that such a situation is occuring requires the agent to infer the soldier's plans (and deviations from them), and that of the enemy (or other agents in the environment that contribute to the situation), from their sequence of actions. This can be framed as the task of *plan recognition* (Kautz and Allen 1986). One approach to plan recognition is case-based plan recognition (CBPR) (Cox and Kerkez 2006). Using a case-based reasoning (CBR) (Lopez de Mantaras et al. 2005) approach to solve this task has the advantage that no model of the robot's teammates is required to recognize the plan being executed and predict their future actions; all that is required is a *plan library* (or *case base*) that enables retrieval of similar plans given similar observed action sequences.

However, CBPR may require a large plan library that corresponds to the large number of situations that the robot might reasonably be expected to encounter. This requires efficient indexing schemes to ensure that plan retrieval time does not increase as a linear function of plan library size. Long runtimes pose an issue when the robot is expected to react quickly to some situations (e.g., enemy fire).

To address this we developed PPC (Plan Projection and Clustering), an algorithm for increasing the runtime speed of CBPR systems. PPC projects plans into a Euclidean space and then hierarchically clusters them. In our empirical analysis, we confirmed that PPC can increase runtime speed while sacrificing only small reductions in plan retrieval accuracy (approximately 4% reduction in accuracy, with an approximately 73% reduction in runtime, in our evaluation).

In the rest of this paper, we describe related work, introduce PPC, describe its empirical study, discuss the results, future work, and conclude.

Related Work

Plan recognition is often conceived of as the inverse of planning. We take inspiration for our approach to increasing CBPR runtime speed from an automated planning perspective. Some automated planning algorithms leverage multiple levels of abstraction; they generate a plan first at an abstract level where planning processes are more tractable, and then refine the generated plan at lower abstraction levels (Sacerdoti 1974; Knoblock 1994). This enables devoting fewer resources to processes at lower levels of abstraction because the space is pruned by processes at higher levels of abstraction. Abstraction makes a plan available for reasoning, including adaptation, at higher abstraction levels. For example, Kambhampati and Hendler (1992) describe how a (nonlinear) planner can accomodate incremental changes in problem specifications (e.g., from a user) through successive adaptations of a hierarchical plan.

Several case-based planning researchers have used multiple abstraction levels to represent plans (Cox, Munoz-Avila, and Bergmann 2005). When solving a new problem, these algorithms typically perform case retrieval first at a highly abstract representation level, and then (if needed) reuse the retrieved plan to constrain search on successively lower abstraction levels. This yields two primary advantages. First, depending on the indexing strategy and how abstractions are encoded, it can permit a single plan to be retrieved and adapted in a larger variety of ways (i.e., increase its coverage) than if it is represented only at its most concrete level. Second, and central to this paper, under some conditions this approach can reduce overall retrieval complexity and time. For example, Smyth and Cunningham (1992) describe how this approach can be applied to a software design task. Bergmann and Wilke (1996) instead focus on a process planning task. They found that abstraction substantially reduced retrieval (and adaptation) time. Branting and Aha (1995) report similar results (for a synthetic planning task), as have several other investigators.

Using hierarchical representations for algorithms related to the k-nearest neighbor classifier has long been a focus in studies on supervised learning, as has been reported in several disciplines. Within the CBR community, Wess et al. (1993) were one of the first groups to contribute to this topic. They use k-d trees to obtain $O(log_2 n)$ average retrieval times, where n is the number of cases. They also introduce several extensions, including virtual bounds to constrain search. They report substantial reductions in retrieval time in comparison to using the original k-d tree algorithm. Daelemans et al. (1997) instead use tries to hierarchically represent cases in natural language processing tasks, and found that it also substantially reduces retrieval time (and storage requirements). (Müller and Bergmann 2014) describe their Hierarchical Bisecting Partitioning Around Medoids algorithm for creating binary cluster trees for use as a fast index structure in process-oriented CBR. Many other researchers have reported similar results.

In contrast, most prior work on CBPR (e.g., Fagan and Cunningham (2003); Cox and Kerkez (2006); Tecuci and Porter (2009); Molineaux et al. (2009)) has not investigated the use of hierarchical indexing techniques. An exception is the recent work by Sánchez-Ruiz and Ontañón (2014), who introduce Least Common Subsumer (LCS) Trees and apply them to induce a hierarchical clustering on cases (i.e., ⟨plan,goal⟩ pairs). Their refinement approach for computing plan similarity, in comparison with other similarity functions, attains a high accuracy for goal prediction while also substantially reducing retrieval time. Our approach instead generates vectors from plans represented as action-state graphs before indexing them hierarchically. Our future work will include an empirical comparison with the LCS Trees method.

Finally, our work in this paper complements our group's earlier work (Vattam, Aha, and Floyd 2014), which did not investigate the use of indexing strategies such as the one we introduce in his paper.

Hierarchy Construction and Plan Retrieval

Representing plans at multiple levels of abstraction to reap CBPR efficiency gains requires selecting a plan transformation/abstraction technique. We propose a domain-independent technique that we expect will work well, to varying degrees, for a variety of plan representations and domains. In this section we describe how PPC constructs plan hierarchies and how plans are subsequently retrieved.

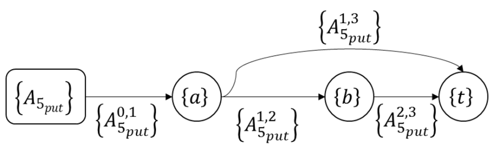

Figure 1: An example predicate encoding graph $\varepsilon(\boldsymbol{p})$ corresponding to $\boldsymbol{p} = put(block : a, block : b, table : t)$

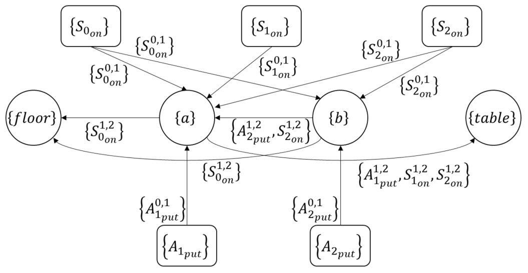

Figure 2: An example of an action state sequence graph $\varepsilon^{\$}$ for $\$ =< (null, \{on(a, floor), on(b, floor)\}), (put(a, table), \{on(a, table)\}), (put(b, a), \{on(a, table), on(b, a)\}) >$

Representation of plans

We model a plan as an *action-state sequence* $\$ = < (\boldsymbol{a_0}, \boldsymbol{s_0}), ..., (\boldsymbol{a_n}, \boldsymbol{s_n}) >$, where each $\boldsymbol{a_i}$ is an action and $\boldsymbol{s_i}$ is the state obtained by executing $\boldsymbol{a_i}$ in $\boldsymbol{s_{i-1}}$. Further, for any $\$$ to be a plan, $\boldsymbol{s_0}$ and $\boldsymbol{s_n}$ must be initial and goal states, and $\boldsymbol{a_0}$ be null. An action $\boldsymbol{a}$ in $(\boldsymbol{a}, \boldsymbol{s}) \in \$$ is a ground literal $\boldsymbol{p} = p(o_1 : t_1, ..., o_n : t_n)$, where $p \in \boldsymbol{P}$ (a finite set of predicate symbols), $o_i \in \boldsymbol{O}$ (a finite set of typed constants representing objects), and t_i is an instance of o_i. A state $\boldsymbol{s}$ in $(\boldsymbol{a}, \boldsymbol{s}) \in \$$ is a set of facts $\{\boldsymbol{p_1}, \boldsymbol{p_2}, ...\}$.

We then encode a plan $\$$ as an *action sequence graph* $\varepsilon^{\$}$. Due to space constraints, here we provide an example of an action sequence graph. (Please see (Vattam, Aha, and Floyd 2014) for additional details.) There are two steps to encode $\$$ as $\varepsilon^{\$}$. First, for each $\boldsymbol{p}$ in $\$$ (be it an action or a state fact) we encode it as a predicate encoding graph $\varepsilon(\boldsymbol{p})$. For example, suppose the predicate $\boldsymbol{p} = put(block : a, block : b, table : t)$ appears as an action in the fifth ($k = 5$) action-state pair of $\$$. The nodes of this predicate are $\{A_{5_{put}}\}$, $\{a\}$, $\{b\}$, and $\{t\}$, and suppose the edges are $[A_{5_{put}}, a]$, $[a, b]$, $[a, t]$, and $[b, t]$, with labels $\{A^{0,1}_{5_{put}}\}$, $\{A^{1,2}_{5_{put}}\}$, $\{A^{1,3}_{5_{put}}\}$, and $\{A^{2,3}_{5_{put}}\}$ respectively. Figure 1 displays the encoding graph for this predicate. If $\boldsymbol{p}$ was a state fact, the A's in the labels would be replaced by S's. Second, we take a union of all the predicate encoding graphs to obtain $\varepsilon^{\$} = \cup_{(a,s)\in\$}(\varepsilon(\boldsymbol{a}) \cup (\cup_{p\in s}\varepsilon(\boldsymbol{p})))$. Figure 2 shows an example of a complete action sequence graph for $\$ = < (null, \{on(a, floor), on(b, floor)\}), (put(a, table), \{on(a, table)\}), (put(b, a), \{on(a, table), on(b, a)\}) >$

For a given plan library L, PPC constructs a plan hierarchy using the following steps:

1. **Create Distance Matrix**: Select and apply a distance

metric d to each pair of plans in L to produce a distance matrix M. The choice of d is a parameter to PPC.

2. **Project Plans**: Project the plans of L into a Euclidean space of N dimensions, where the projection method and N are parameters.
3. **Plan Clustering**: Iteratively cluster the plans in Euclidean space, starting with a single cluster containing all plans. Continue until k clusters are generated. Both the choice of clustering algorithm and k are parameters to PPC.

The induced clusters denote abstractions for the plans they contain in the sense that representing a plan through cluster membership is coarse and concise but captures important information on the nature of the plan as determined by d. This approach may generate non-optimal hierarchies, but its advantage is that it does not depend on in-depth knowledge of the structure of plans in potential queries and in the case base, which may require intimate knowledge of the planner and domain. For example, in our domain, where an agent controls a robot embedded with a human team, the planners are humans and the domain is the physical world, both of which are challenging to model.

PPC performs plan retrieval using the following steps:

1. **Hierarchical Matching**: Recursively match a query (i.e., a given plan) to its closest cluster, continuing until a set of plans at a leaf node is reached.
2. **Concrete-Level Matching**: Return the best-matching plan within this set using the distance metric d used to create M.

Each of PPC's five parameters can be tuned to obtain optimal performance for a given plan library. In the next section we vary N and k, while holding the choices for the projection method, distance metric, and clustering algorithm constant. There will be a trade-off between retrieval accuracy and runtime; what constitutes "optimal" performance will depend upon the desired accuracy/runtime balance.

Empirical Study

We empirically evaluated the performance (i.e., speed and accuracy) of PPC to assess two hypotheses:

H1: PPC's performance should increase when plans are stored with their state information.

H2: PPC's performance should increase when there are groups of plans in the plan library (i.e., where plans in a group have similar start and end states).

H1 is worth investigating because state information (i.e., other than action sequences and action arguments) is often readily available, and can in principle be used to improve performance beyond approaches that use only action information. Likewise, **H2** is worth investigating as the performance of PPC will depend on the characteristics of the plan library, and we expect that one such relevant characteristic would be the degree to which the library contains distinct groups of plans.

As a baseline, we also report the performance of plan retrieval when given a flat (i.e., non-hierarchical) case base.

Empirical Method

We used two performance metrics: (1) the number of plan comparison operations (i.e., distance metric computations) and (2) accuracy (the proportion of queries for which the correct case was retrieved) . The first metric is a primary factor in the runtime of CBPR systems, while we use the second, rather than precision, due to our testing methodology. In particular, we used a *leave-one-in* strategy (Aha and Breslow 1997), where we sample each plan in the plan library (without removing it), and use it as a query for plan retrieval. Accuracy is then the percentage of plans that are themselves retrieved when used as a query.[1] We repeated this process 20 times, and calculated the average number of plan comparisons and average accuracy for different settings of N and k, as described below.

We tested our hypotheses by applying PPC to plan libraries generated from scenarios using, for this initial study, the blocks world domain. We generated 8 plan libraries, each of which includes 60 plans. Plans were generated using PyHop (Nau 2013), a Python implementation similar to SHOP2 (Nau et al. 2003) with a model of the blocks world domain. We evaluated performance over plans involving block sets of two different sizes to determine the effect that the complexity of the domain and its associated plans would have on PPC. In particular, our first set of four libraries involve scenarios that contain 8 blocks, while the latter set of four libraries involve scenarios that contain 26 blocks.

The plan libraries within each set differ according to two conditions. The first condition concerns whether the plans include state information, which allows us to test **H1** (i.e., plans without state information consist of only action sequences and action arguments). To test **H2**, we compared performance between plans involving random states and plans involving ordered states to observe the effect that this has on PPC's performance. The plans of the *unstructured* libraries, involving random states, were produced by, starting with no blocks on the table, iteratively placing blocks on either the table or on another free block until all blocks were placed. The plans of the *structured* libraries, involving non-random block configurations, were produced by selecting one of three start states and one of two goals states, applying perturbations to those states, and generating a plan to go from start to goal state.

PPC has five parameters, as mentioned previously. First, the distance metric d we used in our experiments is Johnson's (1985) similarity metric, which is defined as follows: Let G_1 and G_2 be the action-sequence graphs of two plans being compared. The set of vertices in each graph is divided into l partitions by label type, and then sorted in a non-increasing total order by degree. Let L_1^i and L_2^i denote

[1]PPC's accuracy can be less than 100% because retrieval uses cluster centroids as a guide, and a query is not guaranteed to be assigned to the cluster that contains its most similar case. To illustrate, consider two clusters: $\{1, 2, 3\}$ and $\{3.1, 3.1, 3.1\}$, with centroids 2 and 3.1, respectively. A query "3" will be assigned to the second cluster (because 3 is closer to centroid 3.1 than to centroid 2) even though it is contained within the first cluster.

the sorted degree sequences of a partition i in the action-sequence graphs G_1 and G_2, respectively. An upper bound on the number of vertices $V(G_1, G_2)$ and edges $E(G_1, G_2)$ of the MCS of these two graphs can then be computed as:

$$|mcs(G_1, G_2)| = V(G_1, G_2) + E(G_1, G_2)$$

where:

$$V(G_1, G_2) = \sum_{i=1}^{l} min(|L_1^i|, |L_2^i|), E(G_1, G_2)$$

$$= \left\lfloor \sum_{i=1}^{l} \sum_{j=1}^{min(|L_1^i|,|L_2^i|)} min(|E(v_1^{i,j})|, |E(v_2^{i,j})|)/2 \right\rfloor$$

and where $v_1^{i,j}$ denotes the j^{th} vertex of the L_1^i sorted degree sequence, and $E(v_1^{i,j})$ denotes the set of edges connected to vertex $v_1^{i,j}$. Johnson's similarity is given by: $sim(G_1, G_2) = (|mcs(G_1, G_2)|)^2/(|G_1||G_2|)$. (Please see Vattam et al. (2014) for an example similarity calculation of two graphs using this metric.)

Second, we used multidimensional scaling (MDS) (Kruskal 1964) to project plans. It takes as input a set of distances between entities, as represented in a distance matrix, and projects them into Euclidean space such that the distance between points, associated with plans, is preserved with a certain tolerance (a given set of distance relations will not necessarily be precisely expressible using a set of points in N-dimensional Euclidean space). We set $N \in [2, 9]$ but, due to space constraints, will discuss only a sample of these results.

Finally, for PPC's clustering method, we used k-means and set $k \in [2, 6]$ to produce a hierarchy. We selected k-means because it is well known, works well, and is fast.

PPC starts with all plans in one cluster, and divides this into sub-clusters to produce one level of the hierarchy. It then recurses on each sub-cluster until a desired depth is attained. Each leaf of the hierarchy will consist of a set of plans rather than a set of cluster centers. For each cluster the plan closest to the cluster center was selected to represent it. When matching a query to a cluster, it is compared against that cluster's representative plan using distance metric d. Querying the case base thus involves only distances as produced by d between plans; the query is not projected into the same Euclidean spaced used to construct the hierarchy. Partly due to the simplicity of the datasets, we set the number of levels in our abstraction hierarchies to 2 (i.e., one level of clusters and one level of ground plans). Additional layers would be of greater use on more structured and differentiable datasets.

Results

We applied PPC to the 8 plan libraries described in the preceding section. We denote these libraries using $L_{n,l,i}$, where $n \in \{8, 26\}$ refers to the number of blocks used, $l \in \{s, u\}$ refers to whether the library's plans are structured or unstructured, and $i \in \{t, f\}$ indicates whether state information was included with these plans.

Table 1 displays the results, when $n = 8$ (i.e., scenarios with 8 blocks), for the baseline (i.e., no clustering) and for one setting of PPC, namely when $N = 9$ and $k = 2$. Table 2 displays similar results for when $n = 26$, this time when $N = 9$ and $k = 6$. These parameter values were representative of PPC's results. When using a flat library, imperfect retrieval accuracies occurred because Johnson's similarity metric could not always distinguish between correct and similar matches.

Table 1: Average number of plan comparisons (#PC) and retrieval accuracy (ACC) for plans whose scenarios contain 8 blocks. The results are shown for a Flat library and for PPC when $N = 9$ and $k = 2$.

	Flat		PPC$_{9,2}$	
Library	#PC	ACC	#PC	ACC
L$_{8,u,t}$	60	1.00	33.55	0.96
L$_{8,u,f}$	60	0.95	39.76	0.79
L$_{8,s,t}$	60	0.75	32.53	0.73
L$_{8,s,f}$	60	0.42	31.68	0.32

Table 2: Same as Table 1, but where scenarios contain 26 blocks, and for PPC $N = 9$ and $k = 6$.

	Flat		PPC$_{9,6}$	
Library	#PC	ACC	#PC	ACC
L$_{26,u,t}$	60	1.00	16.08	0.31
L$_{26,u,f}$	60	1.00	15.92	0.32
L$_{26,s,t}$	60	1.00	16.17	0.96
L$_{26,s,f}$	60	0.68	16.34	0.59

PPC's average accuracy for $L_{26,s,t}$ (Table 2) was higher than both $L_{26,s,f}$ and $L_{26,u,t}$; this supports both **H1** and **H2**. However, while the accuracy of $L_{8,s,t}$ (Table 1) was higher than $L_{8,s,f}$, it was not higher than $L_{8,u,t}$. Thus, this supports **H1** but not **H2**.

Tight groupings of plans can improve the quality of clustering and plan retrieval. However, it can also complicate matching the query to a plan, given the similarity of plans within each group. The reason that average accuracy is lower for $L_{8,s,t}$ than for $L_{8,u,t}$, both when using a flat plan library and when using PPC, is that the similarity between plans within a group is too high to easily distinguish them. We surmise that PPC needs more than 8 blocks to increase recognition accuracy for libraries with "structure".

The low accuracies of $L_{26,u,t}$ and $L_{26,u,f}$ in comparison to $L_{8,u,t}$ and $L_{8,u,f}$ is largely due to the different k values (see Figure 5 for the relation of k and accuracy for $L_{26,u,t}$).

Figures 3 and 4 provide more detail concerning the utility of including state information in PPC's plan representation; it compares the accuracy of $L_{26,s,t}$ and $L_{26,s,f}$ for various values of N when $k = 6$. As shown, the accuracy of $L_{26,s,t}$ exceeds that of $L_{26,s,f}$, supporting **H1**. For $N = 9$, this difference is statistically significant according to Welch's t-test ($p < 0.01$).

Similarly, Figures 3 and 5 provide more detail on PPC's performance when the plan library is structured; it compares the accuracy of $L_{26,s,t}$ and $L_{26,u,t}$ for the same values for N and k. As shown, the accuracy of $L_{26,s,t}$ exceeds that of $L_{26,u,t}$, supporting **H2**. For $N = 9$ and $k = 6$, this difference is again statistically significant ($p < 0.01$).

In summary, these results indicate that PPC's recognition accuracy increases when state information is included with

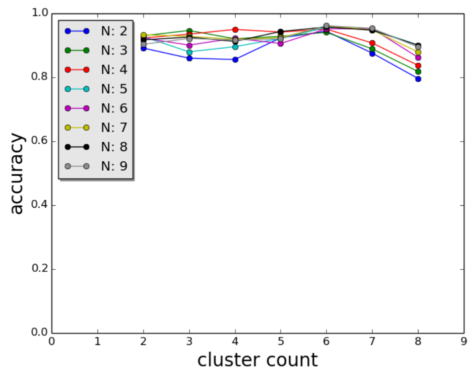

Figure 3: PPC's average accuracy for $L_{26,s,t}$ when varying the number of MDS-projected dimensions.

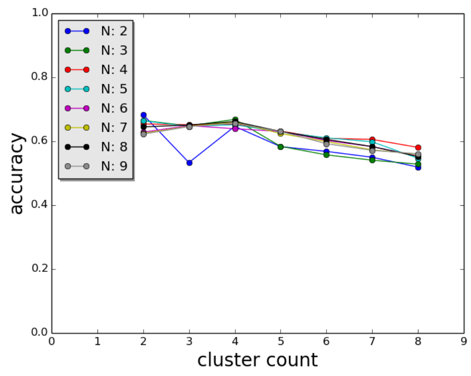

Figure 4: PPC's average accuracy for $L_{26,s,f}$ when varying the number of MDS-projected dimensions.

the stored plans. Furthermore, for libraries whose plans are grouped, if the domain is sufficiently rich such that the members of each group are distinguishable (by the similarity metric), then PPC's average recognition accuracy will be higher than for similar, but ”unstructured”, plan libraries.

Discussion

Our study showed that PPC speeds up plan retrieval with only small reductions to accuracy (for plan libraries that are structured and include state information, in accordance with **H1** and **H2**) with no knowledge of either the planner or the domain. Algorithms that are more effective at reducing runtime while maintaining accuracy may use heuristics garnered from detailed knowledge of the task model of the agents and the domain in which they act. However, this is not always feasible, given the knowledge engineering resources available. For example, recognizing the plans of other agents in an open environment such as the physical world is difficult. Thus, there is a need for a general algorithm such as PPC to reduce the runtime of CBPR systems.

We expect that accuracy will be lower without state information, which can help to distinguish plans. Accuracy is sometimes lower for structured plan libraries. We conjecture

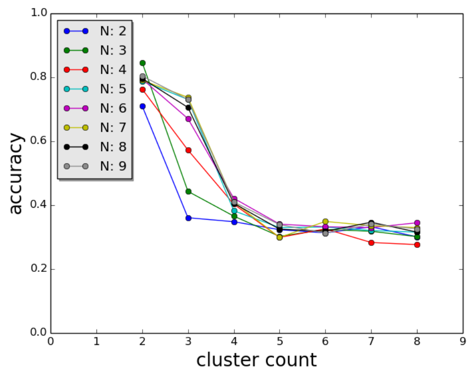

Figure 5: PPC's average accuracy for $L_{26,u,t}$ when varying the number of MDS-projected dimensions.

that this is because structured plan libraries contain sets of plans with higher intra-group similarity.

Future Work

We used leave-one-in experiments and did not test PPC's ability when queried with a *novel* plan. To do so, we will use PPC for SET-PR, a plan recognizer for controlling an unmanned ground robot that is assisting a detached reconnaissance team. SET-PR is designed for domains with partial observabilty and noisy actions (Vattam et al. 2014). We will test PPC's ability to retrieve plans for SET-PR in real-time simulations, measuring its speed and SET-PR's perfomance. We will also empirically compare PPC versus Sánchez-Ruiz and Ontañón's (2014) LCS Trees algorithm. Also, we will examine the use of multiple abstraction hierarchies associated with different plan lengths. This would be desirable if the chosen distance metric could not effectively compare plans of different lengths, as is true for Johnson's distance metric.

Representing plans as points in Euclidean space allows PPC to be used with any clustering method that use Euclidean spatial representations. Some clustering algorithms operate directly over a similarity matrix, making projection into Euclidean space unnecessary. Spectral clustering methods can operate over a similarity matrix. K-medoids can as well and has the advantage that each cluster is associated with a single data-point (which would be its representative plan). Future work includes comparing the performance across different clustering methods, including those that do not require projecting into Euclidean space.

Conclusion

Case-based plan recognizers have usually been investigated with relatively small plan libraries, which are amenable for fast plan retrieval. However, real-world environments may require fast retrieval from much larger libraries. We described PPC, an algorithm that creates an abstraction hierarchy to index plans. During plan retrieval it traverses down this hierarchy and searches only the subset of plans associated with the matched cluster in a leaf. The hierarchy is created by projecting plans into a Euclidean space and then peforming iterative clustering.

We found that PPC increased runtime speed by a factor greater than 3. When applied to a plan library containing both state information and structure, PPC recorded only relatively small accuracy degradations. We examined two hypotheses: **H1** (i.e., PPC's performance increases when plans are stored with state information) and **H2** (i.e., PPC's performance is higher for plan libraries whose plans are sampled from groups). We found support for **H1** and **H2** when using 26 blocks. When using 8 blocks, **H1** was supported, but not **H2**. This is likely because 8 blocks does not provide a textured enough domain for plans within groupings to be easily differentiable. A key property of PPC is that it achieves this reduction in the number of plan comparisons without requiring domain- and agent-specific heuristics, whose creation is dependent on the skill of a knowledge engineer.

Acknowledgements

Thanks to OSD ASD (R&E) for sponsoring this research. The first author was supported as a 2014 NREIP intern at NRL while conducting this research. The views and opinions contained in this paper are those of the authors and should not be interpreted as representing the official views or policies, either expressed or implied, of NRL or OSD.

References

Aha, D. W., and Breslow, L. A. 1997. Refining conversational case libraries. In *Proceedings of the Second International Conference on Case-Based Reasoning*, 267–278. Springer.

Bergmann, R., and Wilke, W. 1996. On the role of abstraction in case-based reasoning. In *Proceedings of the Third European Workshop on Case-Based Reasoning*, 28–43. Springer.

Branting, L. K., and Aha, D. W. 1995. Stratified case-based reasoning: Reusing hierarchical problem solving episodes. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence*, 384–390. Morgan Kaufmann.

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

Cox, M. T.; Munoz-Avila, H.; and Bergmann, R. 2005. Case-based planning. *Knowledge Engineering Review* 20:283–287.

Daelemans, W.; van den Bosch, A.; and Weijters, T. 1997. Igtree: Using trees for compression and classification in lazy learning algorithms. *Artificial Intelligence Review* 11:407–432.

Fagan, M., and Cunningham, P. 2003. Case-based plan recognition in computer games. In *Proceedings of the Fifth International Conference on Case-Based Reasoning*, 161–170. Springer.

Johnson, M. 1985. Relating metrics, lines and variables defined on graphs to problems in medicinal chemistry. In Alavi, Y.; Chartrand, G.; Lick, D.; Wall, C.; and Lesniak, L., eds., *Graph theory with applications to algorithms and computer science*. John Wiley & Sons.

Kambhamapti, S., and Hendler, J. A. 1992. A validation-structure-based theory of plan modifications. *Artificial Intelligence* 55:193–258.

Kautz, H., and Allen, J. F. 1986. Generalized plan recognition. In *Proceedings of the Fifth National Conference on Artificial Intelligence*, 32–38. Morgan Kaufmann.

Knoblock, C. 1994. Automatically generating abstractions for planning. *Artificial Intelligence* 68:243–302.

Kruskal, J. 1964. Multidimensional scaling by optimizing goodness of fit to a nonmetric hypothesis. *Psychometrika* 29(1):1–27.

Lopez de Mantaras, R.; McSherry, D.; Bridge, D.; Leake, D.; Smyth, B.; Craw, S.; Faltings, B.; Maher, M. L.; Cox, M. T.; Forbus, K.; et al. 2005. Retrieval, reuse, revision and retention in case-based reasoning. *Knowledge Engineering Review* 20(03):215–240.

Molineaux, M.; Aha, D. W.; and Sukthankar, G. 2009. Beating the defense: Using plan recognition to inform learning agents. In *Proceedings of the Twenty-Second International FLAIRS Conference*, 337–343. AAAI Press.

Müller, G., and Bergmann, R. 2014. A cluster-based approach to improve similarity-based retrieval for process-oriented case-based reasoning. In *Proceedings of the Twentieth European Conference on Artificial Intelligence*, 639–644. IOS Press.

Nau, D. S.; Au, T.-C.; Ilghami, O.; Kuter, U.; Murdock, J. W.; Wu, D.; and Yaman, F. 2003. Shop2: An htn planning system. *Journal of Artificial Intelligence Research* 20:379–404.

Nau, D. 2013. Game applications of HTN planning with state variables. In Buro, M.; Éric Jacopin; and Vassos, S., eds., *Planning in Games: Papers from the ICAPS Workshop*.

Sacerdoti, E. D. 1974. Planning in a hierarchy of abstraction spaces. *Artificial intelligence* 5:115–135.

Sanchez-Ruiz, A. A., and Ontanon, S. 2014. Least common subsumer trees for plan retrieval. In *Proceedings of the Twenty-Second International Conference on Case-Based Reasoning*, 405–419. Springer.

Smyth, B., and Cunningham, P. 1992. Deja vu: A hierarchical case-based reasoning system for software design. In *Proceedings of the Tenth European Conference on Artificial Intelligence*, 587–589. Wiley and Sons.

Tecuci, D., and Porter, B. W. 2009. Memory based goal schema recognition. In *Proceedings of the Twenty-Second International FLAIRS Conference*, 111–116. AAAI Press.

Vattam, S.; Aha, D. W.; and Floyd, M. 2014. Case-based plan recognition using action sequence graphs. In *Proceedings of the Twenty-Second International Conference on Case-Based Reasoning*, 495–510. Springer.

Wess, S.; Althoff, K.-D.; and Derwand, G. 1993. Using k-d trees to improve the retrieval step in case-based reasoning. In *First European Workshop on Case-Based Reasoning*, 167–181. Springer.