Abductive Plan Recognition by Extending Bayesian Logic Programs

Sindhu Raghavan and Raymond J. Mooney

Department of Computer Science, University of Texas,
1616 Guadalupe, Suite 2.408, Austin TX 78701, USA
{sindhu,mooney}@cs.utexas.edu

Abstract. Plan recognition is the task of predicting an agent's top-level plans based on its observed actions. It is an abductive reasoning task that involves inferring cause from effect. Most existing approaches to plan recognition use either first-order logic or probabilistic graphical models. While the former cannot handle uncertainty, the latter cannot handle structured representations. In order to overcome these limitations, we develop an approach to plan recognition using Bayesian Logic Programs (BLPs), which combine first-order logic and Bayesian networks. Since BLPs employ logical deduction to construct the networks, they cannot be used effectively for plan recognition. Therefore, we extend BLPs to use logical abduction to construct Bayesian networks and call the resulting model Bayesian Abductive Logic Programs (BALPs). We learn the parameters in BALPs using the Expectation Maximization algorithm adapted for BLPs. Finally, we present an experimental evaluation of BALPs on three benchmark data sets and compare its performance with the state-of-the-art for plan recognition.

1 Introduction

Plan recognition is the task of predicting an agent's top-level plans based on its observed actions. It is an abductive reasoning task that involves inferring cause from effect [8]. Traditionally, plan-recognition approaches have been based on first-order logic in which a knowledge-base of plans and actions is developed for the domain and then default reasoning [15] or logical abduction [24] is used to predict the best plan based on the observed actions. However, these approaches are unable to handle uncertainty in the observations or background knowledge and are incapable of estimating the likelihood of different plans. An alternative approach to plan recognition is to use probabilistic methods such as Abstract Hidden Markov Models [5] or statistical n-gram models [2]. While these approaches handle uncertainty, they cannot handle structured representations as they are essentially propositional in nature. As a result, it is also difficult to incorporate planning domain knowledge in these approaches.

The main focus of this paper is to develop an approach to plan recognition that overcomes the limitations described above. Recently, a number of formalisms that integrate both first-order logic and probabilistic graphical models have been developed in the area of *statistical relational learning* (SRL) [11]. Since these formalisms combine the strengths of both approaches, they are well suited for solving problems like plan recognition. We explore the application of one such formalism to plan recognition.

Of the various SRL formalisms that have been developed, Markov Logic Networks (MLNs), [29], which combine first-order logic and undirected graphical models (Markov nets) have been used for abductive plan recognition by Kate and Mooney [14]. Since MLNs employ deduction for logical inference, they adapt MLNs for abduction by adding reverse implications for every rule in the knowledge base. However, the addition of these rules increases the size and complexity of the MLN, resulting in a computationally expensive model.

In this paper, we explore the application of Bayesian Logic Programs (BLPs) [16], which combine first-order Horn logic and Bayesian networks to plan recognition. BLPs use SLD resolution to generate proof trees, which are then used to construct a ground Bayes net for a given query. However, deduction is unable to construct proofs for abductive problems such as plan recognition. Therefore, we extend BLPs to use logical abduction to construct proofs. In logical abduction, missing facts are assumed when necessary to complete proof trees, and we use the resulting abductive proof trees to construct Bayes nets. We call the resulting model Bayesian Abductive Logic Programs (BALPs). Like all SRL formalisms, BALPs combine the strengths of both first-order logic and probabilistic graphical models, thereby overcoming the limitations of traditional plan recognition approaches mentioned above.

First, we present the necessary enhancements to BLPs to support abduction. Next, we discuss how to learn the parameters in BALPs using the Expectation Maximization algorithm adapted for BLPs [18]. Finally, we present an experimental evaluation of BALPs on three benchmark data sets for plan-recognition and compare its performance with the state-of-the-art.

2 Background

2.1 Logical Abduction

In a logical framework, abduction, is usually defined as follows [28]:

- **Given:** Background knowledge B and observations O, both represented as sets of formulae in first-order logic, where B is typically restricted to a set of Horn clauses and O to a conjunction of ground literals.
- **Find:** A hypothesis H, also a set of logical formulae, such that $B \cup H \not\models \bot$ and $B \cup H \models O$.

Here $\models$ stands for logical entailment and $\bot$ for false, i.e. find a set of assumptions that is consistent with the background theory and explains the observations. There are generally many hypotheses H that explain a particular set of observations O. Following Occam's Razor, the best hypothesis is typically selected based on minimizing $|H|$.

2.2 Bayesian Logic Programs

Bayesian logic programs (BLPs) [16] can be viewed as templates for constructing *directed* graphical models (Bayes nets). Given a knowledge base as a special kind of logic program, standard logical deduction (SLD resolution) is used to automatically construct a Bayes net for a given problem. More specifically, given a set of facts and a query, all possible Horn-clause proofs of the query are constructed and used to build a Bayes net for answering the query. Standard probabilistic inference techniques are then used to compute the most probable answer.

More formally, a BLP consists of a set of *Bayesian clauses*, definite clauses of the form $A|A_1, A_2, A_3, ...A_n$, where $n \geq 0$ and A, A_1, A_2, A_3,......,A_n are *Bayesian predicates* (defined below). A is called the head of the clause ($head(c)$) and (A_1, A_2, A_3,....,A_n) is the body ($body(c)$). When $n = 0$, a Bayesian clause is a fact. Each Bayesian clause c is assumed to be universally quantified and range restricted, i.e $variables\{head\} \subseteq variables\{body\}$, and has an associated *conditional probability distribution*: $cpd(c) = P(head(c)|body(c))$.

A *Bayesian predicate* is a predicate with a finite domain, and each ground atom for a Bayesian predicate represents a random variable. Associated with each Bayesian predicate is a combining rule such as *noisy-or* or *noisy-and* that maps a finite set of *cpd*s into a single *cpd* [26]. Let A be a Bayesian predicate defined by two Bayesian clauses, $A|A_1, A_2, A_3, ...A_n$ and $A|B_1, B_2, B_3, ...B_n$, where cpd_1 and cpd_2 are their cpd's. Let θ be a substitution that satisfies both clauses. Then, in the constructed Bayes net, directed edges are added from the nodes for each $A_i\theta$ and $B_i\theta$ to the node for $A\theta$. The combining rule for A is used to construct a single cpd for $A\theta$ from cpd_1 and cpd_2. The probability of a joint assignment of truth values to the final set of ground propositions is then defined in the standard way for a Bayes net: P(X) = $\prod_i P(X_i|Pa(X_i))$, where $X = X_1, X_2, ..., X_n$ represents the set of random variables in the network and $Pa(X_i)$ represents the parents of X_i. The *cpd*s for Bayesian clauses can be learned using the methods described by Kersting and De Raedt [18]. Once a ground network is constructed, standard probabilistic inference methods can be used to answer various types of queries [20].

3 Bayesian Abductive Logic Programs

Bayesian Abductive Logic Programs (BALPs) are an extension of BLPs. In plan recognition, the known facts are insufficient to support the derivation of deductive proof trees for the requisite queries. By using *ab*duction, missing literals can be assumed in order to complete the proof trees needed to determine the structure of the ground network. We first describe the abductive inference procedure used in BALPs. Next we describe how probabilistic parameters are specified and how probabilistic inference is performed. Finally, we discuss how parameters can be automatically learned from data.

3.1 Logical Abduction

Let O_1, O_2, ..., O_n be the set of observations. We derive a set of most-specific abductive proof trees for these observations using the method originally proposed by Stickel [32]. The abductive proofs for each observation literal are computed by backchaining on each O_i until every literal in the proof is proven or assumed. A literal is said to be proven if it unifies with some fact or the head of some rule in the knowledge base, otherwise it is said to be assumed. Since multiple plans/actions could generate the same observation, an observation literal could unify with the head of multiple rules in the knowledge base. For such a literal, we compute alternative abductive proofs. The resulting abductive proof trees are then used to build the structure of the Bayes net using the standard approach for BLPs.

Algorithm 1 AbductionBALP

```
Inputs: Background knowledge $KB$ and observations $O_1$, $O_2$, $O_3$, ...., $O_n$ both represented as sets of formulae in first-order logic, where $KB$ is typically restricted to a set of Horn clauses and each $O_i$ is a ground literal.
Output: Abductive proofs for all $O_i$.
1: Let $Q$ be a queue of unproven atoms, initialized with $O_i$
2: while $Q$ not empty do
3:   $A_i \leftarrow$ Remove atom from $Q$
4:   for each rule $R_i$ in $KB$ do
5:     $consequent \leftarrow$ Head literal of $R_i$
6:     if $A_i$ unifies with $consequent$ then
7:       $S_i \leftarrow$ unify $A_i$ and $consequent$ and return substitution
8:       Replace variables in the body of $R_i$ with bindings in $S_i$. Each literal in the
         body of $R_i$ is a new subgoal.
9:       for each $literal_i$ in body of $R_i$ do
10:        if $literal_i$ unifies with head of some rule $R_j$ in $KB$ then
11:          add $literal_i$ to $Q$
12:        else if $literal_i$ unifies with an existing fact then
13:          Unify and consider the literal to be proved
14:        else
15:          if $literal_i$ unifies with an existing assumption then
16:            Unify and use the assumption
17:          else
18:            Assume $literal_i$ by replacing any unbound variables that are exis-
               tentially quantified in $literal_i$ with new Skolem constants.
19:          end if
20:        end if
21:      end for
22:    end if
23:  end for
24: end while
```

The basic algorithm to construct abductive proofs is given in Algorithm 1. The algorithm takes as input a knowledge base (KB) in the form of Horn clauses and a set of observations as ground facts. It outputs a set of abductive proof trees by performing logical abduction on the observations. These proof trees are then used to construct the Bayesian network. For each observation O_i, AbductionBALP searches for rules whose consequents unify with O_i. For each such rule, it computes the substitution from the unification process and substitutes variables in the body of the rule with bindings from the substitution. The literals in the body now become new subgoals in the inference process. If these new subgoals cannot be proved, i.e if they cannot unify with existing facts or with the consequent of any rule in the KB, then they are assumed. In order to minimize the number of assumptions, the assumed literals are first matched with existing assumptions. If no such assumption exists, then any unbound variables in the literal that are existentially quantified are replaced by Skolem constants.

In SLD resolution, which is used in BLPs, if any subgoal literal cannot be proven, the proof fails. However, in BALPs, we assume such literals and allow proofs to proceed till completion. Note that there could be multiple existing assumptions that could unify with subgoals in Step 15. However, if we used all ground assumptions that could unify with a literal, then the size of the ground network would grow exponentially, making probabilistic inference intractable. In order to limit the size of the ground network, we unify subgoals with assumptions in a greedy manner, i.e when multiple assumptions match with a subgoal, we just randomly pick one of them and do not pursue the others. We found that this approach worked well for plan-recognition. For other tasks, domain-specific heuristics could potentially be used to reduce the size of the network.

```
(a) Partial Knowledge Base:
# Shopping
1. inst(?g,going) | inst(?b,shopping), go-step(?b,?g).
2. inst(?sp,shopping-place) | inst(?s,shopping), store(?s,?sp).
# Robbing
3. inst(?p,going) | inst(?r,robbing), go-step(?r,?p).

(b) Observations:
inst(go1,going)
inst(store1,shopping-place)

(c) Ground Abductive Clauses:
inst(go1,going) | inst(a1,shopping), go-step(a1,go1).
inst(go1,going) | inst(a1,robbing), go-step(a1,go1).
inst(store1,shopping-place) | inst(a1,shopping), store(a1,store1).
```

Fig. 1: (a) A partial knowledge base from the Story Understanding data set. All variables start with "?". (b) The logical representation of the observations. (c) The set of ground rules obtained from logical abduction.

We now illustrate the abductive inference process with a simple example from the Story-Understanding benchmark data set described in Section 4.1. Consider the partial knowledge base and set of observations given in Figure 1a and Figure 1b respectively. There are two top-level plans, shopping and robbing, in the knowledge base. Note that the action literal "inst(?g, going)" could be observed as part of both shopping and robbing. For each observation literal in Figure 1b, we recursively backchain to generate abductive proof trees. When we backchain on the literal *inst(go1,going)* using Rule 1, we obtain the subgoals *inst(?b,shopping)* and *go-step(?b,go1)*. These subgoals become

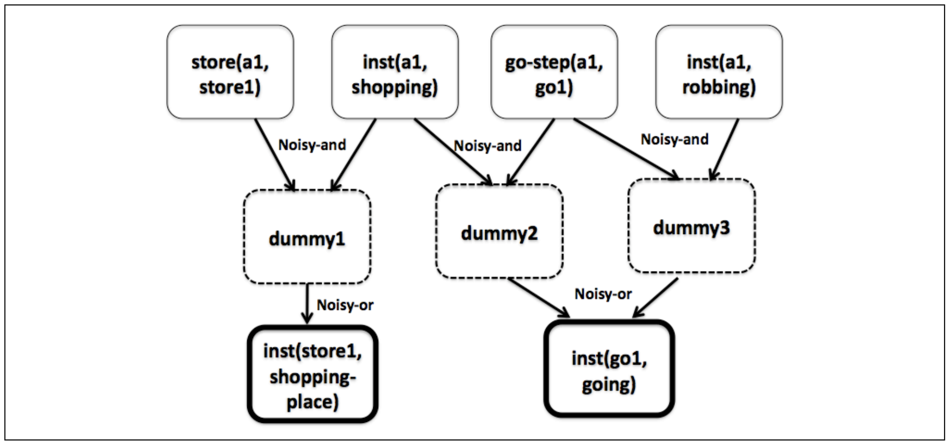

Fig. 2: Bayesian network constructed for example in Figure 1. The nodes with thick borders represent observed actions, the nodes with dotted borders represent intermediate nodes used to combine the conjuncts in the body of a clause, and the nodes with thin borders represent plan literals.

assumptions since no observations or heads of clauses unify with them. Since $?b$ is an existentially quantified variable, we replace it with a Skolem constant $a1$ to obtain the ground assumptions *inst(a1,shopping)* and *go-step(a1,go1)*. We then backchain on literal *inst(go1,going)* using Rule 3 to get subgoals *inst(?r,robbing)* and *go-step(?r,go1)*. We cannot unify *inst(?r, robbing)* with any observation or existing assumptions; however, we can unify *go-step(?r,go1)* with an existing assumption *go-step(a1,go1)*, thereby binding ?r to a1. In order to minimize the number of assumptions, we first try to match literals with unbound variables to existing assumptions, rather than instantiating them with new Skolem constants. Finally, we backchain on the literal *inst(store1,shopping-place)* using Rule 2 to get subgoals *inst(?s,shopping)*, *store(?s,store1)*. Here again, we match *inst(?s, shopping)* to an existing assumption *inst(a1,shopping)*, thereby binding ?s to a1.

Figure 1c gives the final set of ground rules generated by abductive inference. After generating all abductive proofs for all observation literals, we construct a Bayesian network. Figure 2 shows the Bayesian network constructed for the example in Figure 1. Note that since there are no observations/facts that unify with the subgoals (*inst(?b,shopping)*, *go-step(?b,?g)*, *inst(?r,robbing)*, *go-step(?r,?p)*, and *store(?s,?sp)*) generated during backchaining on observations, SLD resolution will fail to generate proofs. This is typical in plan recognition, and as a result, we cannot use BLPs for such tasks.

The only difference between BALPs and BLPs lies in the logical inference procedure used to construct proofs. Once the abductive proofs are generated, BALPs use the same procedure as BLPs to construct the Bayesian network. We further show in Section 3.3 and Section 4.3 that techniques developed for BLPs for learning parameters can also be used for BALPs.

3.2 Probabilistic Parameters and Inference

We now discuss how parameters are specified in BALPs. We use noisy/logical-and and noisy-or models to specify the *cpd*s in the ground Bayesian network as these models compactly encode the *cpd* with fewer parameters, i.e. just one parameter for each parent node. Depending on the domain, we use either a strict *logical-and* or a softer *noisy-and* model to specify the *cpd* for combining evidence from the conjuncts in the body of a clause. We use a noisy-or model to specify the *cpd* for combining the disjunctive contributions from different ground clauses with the same head. Figure 2 shows the noisy-and and noisy-or nodes in the Bayesian network constructed for the example in Figure 1.

Given the constructed Bayesian network and a set of observations, we determine the best explanation using standard methods for computing the Most Probable Explanation (MPE) [26], which determines the joint assignment of values to the unobserved nodes in the network which has the maximum posterior probability given the observations. To compute multiple alternative explanations, we use the k-MPE algorithm [25] as implemented in Elvira [10]. For other types of exact probabilistic inference (marginal and joint) we use Netica,[1] a commercial Bayes-net software package.

When the complexity of the ground network makes exact inference intractable (as in the Monroe dataset described in Sect. 4), we have to resort to approximate inference. Due to the (noisy/logical) *and* and *or* nodes in the network, there are a number of deterministic constraints, i.e. 0 values in the *cpd*s. As a result, generic importance sampling algorithms like likelihood weighting used in Elvira failed to generate sufficient samples. Hence, we used SampleSearch [12], an approximate sampling algorithm specifically designed for graphical models with multiple deterministic constraints.

3.3 Parameter Learning

Learning can be used to automatically set the noisy-or and noisy-and parameters in the model. We learn these parameters using the EM algorithm adapted for BLPs by Kersting and De Raedt [18]. In supervised training data for plan recognition, one typically has evidence for the observed actions and the top-level plans. However, we usually do not have evidence for network nodes corresponding to subgoals, noisy-ors, and noisy/logical-ands. As a result, there are a number of variables in the ground networks which are always hidden, and hence EM is appropriate for learning the requisite parameters from the partially observed training data. We simplify the problem by learning only the noisy-or parameters and using a deterministic logical-and model to combine evidence from the conjuncts in the body of a clause. We use uniform priors for top-level plans unless otherwise mentioned.

4 Experimental Evaluation

Unfortunately, there are very few benchmark datasets or rigorous experimental evaluations of plan recognition. In this section, we evaluate BALPs on three plan-recognition

[1] `http://www.norsys.com/`

datasets that are available. First, we describe experiments to determine if BALPs are more effective for plan recognition than previous approaches. Then, we describe additional experiments evaluating the automatic learning of BALP parameters.

4.1 Datasets

Monroe / Reformulated Monroe The Monroe dataset is an artificially-generated plan-recognition dataset in the emergency response domain by Blaylock and Allen [1]. This domain includes top level plans such as setting up a temporary shelter, clearing a road wreck, and providing medical attention to victims. The task is to infer a single top level plan from a set of observed actions automatically generated by a planner. The planner used is SHOP2 [22] and the domain knowledge is represented as a hierarchical transition network (HTN). We constructed a logical knowledge base representing the domain knowledge encoded in the HTN. We used 1,000 artificially generated examples in our experiments. Each example instantiates one of the 10 top-level plans and contains an average of 10.19 literals describing a sample execution of this plan.

Due to computational complexity, we were unable to compare the performance of BALPs with Kate and Mooney's [14] MLN approach on this domain. Their approach resulted in an MLN with rules containing multiple existentially quantified variables which produced an exponential number of possible groundings, eventually leading to memory overflow. In order to compare BALPs with this MLN approach, we slightly modified the Monroe domain to eliminate this problem without significantly changing the underlying task. The resulting dataset also had 1,000 examples, with an average of 9.7 observations per example. We refer to this dataset as "Reformulated-Monroe."

Linux The Linux dataset is another plan-recognition dataset created by Blaylock and Allen [3]. Human users were asked to perform various tasks in Linux and their commands were recorded. The task is to predict the correct top level plan from the sequence of executed commands. For example, one of the tasks involves finding all files with a given extension. The dataset consists of 19 top level plans and 457 examples, with an average of 6.1 command literals per example. We constructed the background knowledge base for the Linux dataset based on our knowledge of the commands.

Story Understanding We also used a dataset[2] that was previously used to evaluate abductive story understanding systems [24, 6]. In this task, characters' higher-level plans must be inferred from their actions described in a narrative text. A logical representation of the literal meaning of the text is given for each example. A sample story is: "Bill went to the liquor-store. He pointed a gun at the owner." The plans in this dataset include shopping, robbing, restaurant dining, traveling in a vehicle (bus, taxi or plane), partying and jogging. Most narratives involve more than a single plan. This small dataset consists of 25 development examples and 25 test examples each containing an average of 12.6 literals. We used the background knowledge that was initially constructed for

[2] `http://www.cs.utexas.edu/~ml/accel.html`

the ACCEL system [24]. Figure 1a and Figure 1b give a partial knowledge base and a partial set of observations from this data set.

Each of these data sets evaluates a distinct aspect of plan recognition systems. Since the Monroe domain is quite large with numerous subgoals and entities, it tests the ability of a plan-recognition system to scale to large domains. On the other hand, the Linux data set is not that large, but since the data comes from real human users, it is quite noisy. There are several sources of noise including cases in which users claim they have successfully executed a top-level plan when actually they have not [2]. Therefore, this data set tests the robustness of a plan-recognition system to noisy input. Monroe and Linux involve predicting a *single* top-level plan; however, in the Story Understanding domain, most examples have multiple top-level plans. Therefore, this data set tests the ability of a plan-recognition system to identify multiple top-level plans.

4.2 Comparison with Other Approaches

We now present comparisons to three previous approaches to plan-recognition across the different benchmark datasets.

Monroe and Linux We first compared BALPs with Blaylock and Allen's [2] plan-recognition system on both the Monroe and Linux datasets. Their approach learns statistical n-gram models to separately predict plan schemas (i.e. predicates) and their arguments.

We learned the noisy-or parameters for BALPs using the EM algorithm described in Sect. 3.3. We initially set all noisy-or parameters to 0.9, which gave reasonable performance in both domains. We picked a default value of 0.9 for noisy-or parameters based on the intuition that if a parent node is true, then the child node is true with a probability 0.9. We then ran EM with two starting points – random weights and manual weights (0.9). We found that EM initialized with manual weights generally performed the best for both domains, and hence we used this approach for our comparisons. Even though EM is sensitive to starting point, it outperformed other approaches even when initialized with random weights (see Sect. 4.3). Initial experiments found no advantage to using noisy-and instead of logical-and in these domains, so we did not experiment with learning noisy-and parameters.

For Linux, we performed 10-fold cross validation for evaluation and we ran EM until convergence on the training set for each fold. For Monroe, where more data is available, we used 300 examples for training, 200 examples for validation, and the remaining 500 examples for testing. Note that for Monroe, Blaylock and Allen used 4,500 examples for learning parameters. Using 4500 examples for learning BALP parameters results in large training times, and hence we limited to using 300 examples. We ran EM iterations on the training set until the accuracy on the validation set stopped improving. We then used the final learned set of weights to perform plan-recognition on the test set.

For both Monroe and Linux, the plan-recognition task involves inferring a *single* top level plan that best explains the observations. Hence, we computed the marginal probabilities for all ground instantiations of the plan predicates in the network and picked the single plan instantiation with the highest marginal probability.

Due to differences in Blaylock and Allen's experimental methodology and ours, we are only able to directly compare performance using their *convergence score* [2], the fraction of examples for which the correct plan predicate is inferred (ignoring the arguments) when given *all* of the observations.

Table 1 shows the results. BALPs outperform Blaylock and Allen's system on the convergence score in both domains and the difference in the performances was statistically significant ($p < .05$) as determined by unpaired t-test [3]. We treated the convergence score as the mean of a Bernoulli variable (schema prediction event) and computed variance accordingly, and then used the mean, variance, and sample size to perform an unpaired t-test. The convergence score for Blaylock and Allen's system on Monroe is already quite high, leaving little room for improvement. However, BALPs was still able to improve over this score by 4.5%. On the other hand, the baseline convergence score for Linux was fairly low, and BALPs were able to improve the results by a remarkable 29.1%. Despite this improvement, the overall convergence score for Linux is not that high. Noise in the data is one reason for the modest score. Another issue with this data set is the presence of very similar plans, like find-file-by-ext and find-file-by-name. The commands executed by users in these two plans are nearly identical, making it difficult for a plan recognition system to distinguish them [3].

	BALPs	Blaylock and Allen
Monroe	**98.4**	94.2*
Linux	**46.6**	36.1*

Table 1: Convergence scores for BALPs and Blaylock and Allen's system for Monroe and Linux. '*' indicates that the difference is statistically significant.

Reformulated-Monroe We also compared the performance of BALPs with Kate and Mooney's [14] MLN approach on the Reformulated-Monroe dataset. For MLNs, we were unable to effectively learn clause weights on this dataset since it was intractable to run Alchemy's existing weight-learners due to the sizes of the MLN and data. Hence, we manually set the weights using the heuristics described by Kate and Mooney [14]. To ensure a fair comparison, we also used manual instead of learned weights for BALPs. We uniformly set all noisy-or parameters to 0.9 and used logical-and to combine evidence from conjuncts in the body of a clause, since this gave good performance on the original Monroe data.

We used two different metrics to compare the performance of the two approaches – convergence score and accuracy. We compared the inferred plan with the correct plan to compute the accuracy score. When computing accuracy, partial credit was given for predicting the correct plan predicate with only a subset of its correct arguments. A point was rewarded for inferring the correct plan predicate, then, given the correct predicate, an additional point was rewarded for each correct argument. For example, if the correct plan was $plan_1(a_1, a_2)$ and the inferred plan was $plan_1(a_1, a_3)$, the accuracy was 66.67%.

[3] We did not have access to scores for individual examples for Blaylock and Allen's system, hence we performed an *un*paired t-test.

The observation set for this domain includes all actions executed to implement the top level plan. In order to evaluate performance for partially observed plans, we performed plan recognition given only subsets of the complete action sequence. Specifically, we report results after observing the first 25%, 50%, 75%, and 100% of the executed actions. Table 2 shows the results. "Accuracy-n" is the accuracy when given the first $n\%$ of the observations. BALPs consistently outperform the MLN approach on this data set and all differences are statistically significant ($p < .05$) as determined by the Wilcoxon Sign Rank (WSR) test [30]. The convergence score for BALPs demonstrates a large (25.41%) improvement over MLNs. Finally, we would like to note that the computational complexity of the MLN approach prevented us from running it on the Linux dataset.

	Convergence Score	Accuracy-100	Accuracy-75	Accuracy-50	Accuracy-25
BALP	**99.90**	**97.40**	**66.80**	**32.67**	**9.83**
MLN	79.66*	79.20*	40.51*	19.26*	4.10*

Table 2: Comparative results for Reformulated-Monroe, "*" indicates that the difference is statistically significant.

Story Understanding On Story Understanding, we compared the performance of BALPs with the MLN approach of Kate and Mooney [14] and ACCEL [24], a logical-abduction system that uses a metric to guide its search for selecting the best explanation. ACCEL can use two different metrics: *simplicity*, which selects the explanation with the fewest assumptions and *coherence*, which selects the explanation that maximally connects the input observations. This second metric is specifically geared towards text interpretation by measuring *explanatory coherence* [23]. Currently, this bias has not been incorporated in either the BALP or MLN approach.

For BALPs, we were unable to learn useful parameters from just 25 development examples. As a result, we set parameters manually in an attempt to maximize performance on the development set. As before, a uniform value of 0.9 for all noisy-or parameters seemed to work well for this domain. Unlike other domains, using logical-and to combine evidence from conjuncts in the body of a clause did not yield high-quality results. Using noisy-and significantly improved the results; so we used noisy-ands with uniform parameters of 0.9. Here again, the intuition was that if parent node was false or turned off, then the child node would also be false or turned off with a probability 0.9. To disambiguate between conflicting plans, we set different priors for high level plans to maximize performance on the development data. For the MLN approach, we used Kate and Mooney's [14] system with their manually-tuned weights for this dataset.

Since multiple plans are possible in this domain, we computed the most probable explanation (MPE) to infer the best set of plans. We compared the inferred plans with the ground truth to compute *precision*, *recall*, and *F-measure* (the harmonic mean of precision and recall). As before, partial credit was given for predicting the correct plan predicate with some incorrect arguments. The observed literals in this data are already incomplete and do not include all of the actions needed to execute a plan, so they were used as is.

Table 3 shows the results. BALPs performed better than both ACCEL-Simplicity and MLNs. With respect to F-measure, BALPs improved over MLNs by 15.57% and over ACCEL-Simplicity by 33.65%. However, ACCEL-Coherence still performed the best. Since the coherence metric incorporates extra criteria specific to story understanding, this bias would need to be included in the probabilistic models to make them more competitive. However, the coherence metric is specific to narrative interpretation and not applicable to plan recognition in general.

	BALP	MLN	ACCEL-Simplicity	ACCEL-Coherence
Precision	72.07	67.31	66.45	89.39*
Recall	85.57	68.10*	52.32*	89.39
F-measure	78.24	67.70*	58.54*	89.39*

Table 3: Comparative results for Story Understanding, "*" indicates that the difference wrt BALPs is statistically significant.

Overall, BALPs outperformed most existing approaches on the existing benchmark data sets, thus demonstrating that BALPs are a very effective approach to plan recognition.

4.3 Parameter Learning Experiments

We now describe additional experiments that were designed to determine if EM can effectively learn BALP parameters in different plan-recognition domains.

Learning Methodology We used EM as described in Sect. 3.3 to learn noisy-or parameters for the Linux and Monroe domains.[4] We initially set all noisy-or parameters to 0.9. This gives reasonable performance in both domains, so we compare BALPs with learned noisy-or parameters to this default model which we call "Manual-Weights" (MW). For training, we ran EM with two sets of starting parameters – manual weights (0.9) and random values. We call the former "MW-Start" and the latter "Rand-Start". We used the same training and test splits as described in Section 4.2 for both Linux and Monroe. To measure performance, we computed the convergence score and accuracy scores for various levels of observability as described above.

Learning Results Table 4 shows the results for different models on Linux. MW-Start consistently outperforms MW, demonstrating that parameter learning improves the performance of default BALP parameters on the Linux domain. Rand-Start does marginally better than MW for all but the 50% and 25% levels of partial observability. However, it does not perform as well as MW-Start, showing that learning from scratch is somewhat better than using default parameters but not as effective as starting learning from reasonable default values.

[4] We were unable to learn useful parameters for Story Understanding since the mere 25 development examples were insufficient for training.

	Convergence Score	Accuracy-100	Accuracy-75	Accuracy-50	Accuracy-25
MW	39.82	30.41	28.22	21.84	18.34
MW-Start	**46.6***	**36.32***	**34.06***	**25.45***	**19.83***
Rand-Start	41.57	31.4	29.1	20.53	14.55*

Table 4: Results for parameter learning on Linux, "*" indicates that the difference wrt the MW model is statistically significant.

Table 5 shows the results for different models on Monroe. The performance of MW is already so high that there is little room for improvement, at least for the convergence score. As a result, the MW-Start model could not improve substantially over the MW model. The manual parameters seem to be at a (local) optimum, preventing EM from making further improvements on this data. Rand-Start is performing about as well, sometimes a bit better and sometimes a bit worse than MW, demonstrating that starting from random values the system can learn weights that are about as effective as manual weights for this domain. One reason for the high performance of the MW model on Monroe is the lack of ambiguity in the observations, i.e. there are few observed actions that are part of more than one possible plan. Overall, EM was able to automatically learn effective parameters for BALPs.

	Convergence Score	Accuracy-100	Accuracy-75	Accuracy-50	Accuracy-25
MW	98.4	79.16	46.06	20.67	7.2
MW-Start	98.4	79.16	44.63*	20.26	7.33
Rand-Start	98.4	79.86*	44.73*	19.7*	10.46*

Table 5: Results for parameter learning on Monroe, "*" indicates that the difference wrt the MW model is statistically significant.

5 Discussion

We now discuss various aspects of BALPs that may explain their superior performance. As mentioned earlier, Kate and Mooney's MLN approach [14] cannot be applied to large domains like Monroe since the addition of reverse implications results in a computationally expensive model. As opposed to the explosive grounding of rules in MLNs, BALPs use logical abduction in which only those rules that are relevant to the query are included in the ground network. This results in much smaller networks, enabling BALPs to scale well to large domains. Furthermore, the use of logical abduction allows BALPs to use an existing knowledge base that was created for planning without any modification.

When Blaylock and Allen [2] perform instantiated plan recognition, it is done in a pipeline of two separate steps. The first step predicts the plan schema and the second step predicts the arguments given the schema. Unlike their approach, BALPs are able to jointly predict both the plan and its arguments simultaneously. We believe that BALP's ability to perform joint prediction is at least partly responsible for its superior performance. Both BALP and MLN systems use planning knowledge specified in the form of logical clauses, while Blaylock and Allen's system has no access to domain knowledge. We believe that the ability of BALPs to incorporate domain knowledge is also partly responsible for its superior performance.

Blaylock and Allen's system [2] uses 4500 examples to learn reasonable parameters for the Monroe domain. The MLN system by Kate and Mooney is unable to scale effectively to this domain. On the other hand, BALPs learn effective parameters for this domain from only 300 examples, demonstrating that EM can effectively learn parameters given a reasonable number of examples. Except for the Story Understanding data set, the EM algorithm used in BALPs could learn parameters automatically from data. The inability of EM to learn effective parameters for this data set can be attributed to the lack of a sufficient number of examples. Note that Kate and Mooney's MLN approach was also unable to learn reasonable weights for Story Understanding. Also note that it is possible to learn parameters for Reformulated-Monroe using EM, but we deliberately avoided using learning to ensure a fair comparison with MLNs. Overall, the success of EM in the original Monroe and Linux domains demonstrates that our approach can automatically learn accurate parameters from data.

Overall the results demonstrates that our approach to plan recognition using BALPs is very effective, generally outperforming existing approaches on the three extant benchmark data sets. As mentioned earlier, each data set tests a specific aspect of the system, and BALP's superior performance on these data sets demonstrate that it is a robust approach to plan recognition.

6 Related Work

Charniak and Goldman [7, 6] developed an approach to automatically construct Bayesian networks for plan recognition. Their work is similar to BALPs, but special purpose procedures were used to construct the necessary ground networks rather than using a general-purpose probabilistic predicate logic like MLNs, BLPs, or BALPs. Bui [5] has developed an approach for plan recognition based on Abstract Hidden Markov Models, but this approach cannot handle relational data. Several other systems for plan recognition [24, 14, 2] were already discussed in Section 4.2.

Poole [27] has developed a framework for Horn clause abduction and shows it relations to Bayesian networks, Chen *et. al* [9] extend Stochastic Logic Programs [21] to incorporate abduction, and Sato [31] has also developed a probabilistic logic called PRISM that performs abduction. Kimmig et. al [19] have developed a method to construct probabilistic explanations using ProbLog. However, none of these approaches have been evaluated on the task of plan recognition. Kersting and De Raedt [17] discuss the differences between BLPs and many of these formalisms; and BALPs inherit these same differences.

7 Future Work

The current comparison to MLNs uses the method of Kate and Mooney [14] without automatic learning of weights. In the future, we would like to explore more efficient online-weight learning [13] with MLNs and compare their performance to BALPs. Furthermore, Kate and Mooney's approach to incorporating logical abduction in MLNs can be improved (c.f. [4]), so comparing to enhanced MLN approaches is another area of future work. We would also like to explore approaches based on lifted inference, which allow to perform probabilistic inference without having to construct ground networks in the future. As mentioned in Section 6, there are several probabilistic logics like Poole's Horn Abduction, PRISM, and abductive SLPs that can perform abductive reasoning. It would be interesting to apply these frameworks to plan recognition and compare their performance with that of BALPs. In this paper, the background knowledge base was hand-coded; however, we would like to explore techniques that learn abductive knowledge bases automatically from training data.

8 Conclusions

This paper has introduced an approach to plan recognition based on Bayesian Logic Programs (BLPs). We extended BLPs for plan recognition by employing logical abduction to construct Bayesian networks as opposed to the deductive approach currently used in BLPs. We also demonstrated that the model's parameters can be effectively learned using EM. Empirical evaluations on three benchmark data sets demonstrated that the approach generally outperforms the state-of-the-art in plan recognition. We believe that its superior performance is due to its combination of logical abduction, joint probabilistic inference, and incorporation of domain knowledge.

Acknowledgements

We would like to thank Nate Blaylock for sharing Linux and Monroe data sets, Vibhav Gogate for helping us modify SampleSearch algorithm for our experiments, and Parag Singla for his valuable inputs on the paper. This research was funded by MURI ARO grant W911NF-08-1-0242 and Air Force Contract FA8750-09-C-0172 under the DARPA Machine Reading Program. Experiments were run on the Mastodon Cluster, provided by NSF grant EIA-0303609.

References

1. Blaylock, N., Allen, J.: Generating artificial corpora for plan recognition. In: UM-05. Springer (2005)
2. Blaylock, N., Allen, J.: Recognizing instantiated goals using statistical methods. In: G. Kaminka (Ed.), Workshop on MOO-05. pp. 79–86 (2005)
3. Blaylock, N., Allen, J.F.: Statistical goal parameter recognition. In: ICAPS-04. pp. 297–305 (2004)
4. Blythe, J., J. Hobbs, J., Domingos, P., Kate, R., Mooney, R.: Implementing weighted abduction in Markov logic. In: IWCS-11 (Jan 2011)

5. Bui, H.H.: A general model for online probabilistic plan recognition. In: IJCAI-03 (2003)
6. Charniak, E., Goldman, R.: A probabilistic model of plan recognition. In: AAAI-91. pp. 160–165. Anaheim, CA (1991)
7. Charniak, E., Goldman, R.P.: A semantics for probabilistic quantifier-free first-order languages, with particular application to story understanding. In: IJCAI-89. Detroit, MI (1989)
8. Charniak, E., McDermott, D.: Introduction to Artificial Intelligence. ADDISON, Reading, MA (1985)
9. Chen, J., Muggleton, S., Santos, J.: Learning probabilistic logic models from probabilistic examples. Machine Learning 73(1), 55–85 (2008)
10. Elvira-Consortium: Elvira: An environment for probabilistic graphical models. In: Proceedings of the Workshop on Probabilistic Graphical Models. Cuenca, Spain (2002)
11. Getoor, L., Taskar, B. (eds.): Introduction to Statistical Relational Learning. MITP, Cambridge, MA (2007)
12. Gogate, V., Dechter, R.: Samplesearch: A scheme that searches for consistent samples. In: AISTATS-07 (2007)
13. Huynh, T.N., Mooney, R.J.: Online max-margin weight learning with Markov logic networks. In: SDM-11 (2011)
14. Kate, R.J., Mooney, R.J.: Probabilistic abduction using Markov logic networks. In: IJCAI-09 Workshop on Plan, Activity, and Intent Recognition. Pasadena, CA (July 2009)
15. Kautz, H.A., Allen, J.F.: Generalized plan recognition. In: AAAI. pp. 32–37. Philadelphia, PA (1986)
16. Kersting, K., De Raedt, L.: Towards combining inductive logic programming with Bayesian networks. In: ILP-01. pp. 118–131. Strasbourg, France (September 2001)
17. Kersting, K., De Raedt, L.: Bayesian Logic Programming: Theory and Tool. In: Getoor, L., Taskar, B. (eds.) An Introduction to Statistical Relational Learning. MITP, Cambridge, MA (July 2007)
18. Kersting, K., Raedt, L.D.: Basic principles of learning Bayesian logic programs. In: Probabilistic Inductive Logic Programming. pp. 189–221 (2008)
19. Kimmig, A., De Raedt, L., Toivonen, H.: Probabilistic explanationa based learning. In: ECML-07 (2007)
20. Koller, D., Friedman, N.: Probabilistic Graphical Models: Principles and Techniques. MIT Press (2009)
21. Muggleton, S.: Learning structure and parameters of stochastic logic programs. In: ILP-02. pp. 198–206 (2003)
22. Nau, D., Ilghami, O., Kuter, U., Murdock, J.W., Wu, D., Yaman, F.: Shop2: An HTN planning system. Journal of Artificial Intelligence Research 20, 379–404 (2003)
23. Ng, H.T., Mooney, R.J.: The role of coherence in abductive explanation. In: AAAI-90. pp. 337–442. Detroit, MI (July 1990)
24. Ng, H.T., Mooney, R.J.: Abductive plan recognition and diagnosis: A comprehensive empirical evaluation. In: KR-92. pp. 499–508. Cambridge, MA (Oct 1992)
25. Nilsson, D.: An efficient algorithm for finding the M most probable configurations in probabilistic expert systems. Statistics and Computing 8, 159–173 (1998)
26. Pearl, J.: Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference. MKP, CA (1988)
27. Poole, D.: Probabilistic Horn abduction and Bayesian networks. Artificial Intelligence 64, 81–129 (1993)
28. Pople, H.E.: On the mechanization of abductive logic. In: IJCAI-73. pp. 147–152 (1973)
29. Richardson, M., Domingos, P.: Markov logic networks. Machine Learning 62, 107–136 (2006)
30. Rosner, B.: Fundamentals of Biostatistics. Duxbury Press (2005)
31. Sato, T.: A statistical learning method for logic programs with distribution semantics. In: ICLP-95. pp. 715–729. MIT Press (1995)
32. Stickel, M.E.: A Prolog-like inference system for computing minimum-cost abductive explanations in natural-language interpretation. Tech. Rep. Tech. Note 451, SRI International, CA (Sep 1988)