clp(pfd(Y)) : Constraints for Probabilistic Reasoning in Logic Programming

Nicos Angelopoulos

Department of Computer Science, York University, York, YO10 5DD, UK.
`nicos@cs.york.ac.uk`

Abstract. We argue that the clp(X) framework is a suitable vehicle for extending logic programming (LP) with probabilistic reasoning. This paper presents such a generic framework, clp(pfd(Y)), and proposes two promising instances. The first provides a seamless integration of Bayesian Networks, while the second defines distributions over variables and employs conditional constraints over predicates. The generic methodology is based on attaching probability distributions over finite domains. We illustrate computational benefits of this approach by comparing program performances with a clp(fd) program on a cryptographic problem.

1 Introduction

LP has been an integral part of symbolic problem solving. Its success with crisp reasoning makes its extension to deal with statistical reasoning an appealing proposition. To this direction, a number of approaches, which enhance LP with the ability to reason under uncertainty, have been suggested. These have been mainly based on Probability Theory. Invariably, a probabilistic measure, or interval, is attached to LP clauses or specialised facts.

Such approaches are either based on Markov process principles [9, 4, 7] and subordinate probabilistic reasoning to logical reasoning, or based on Bayesian principles and subordinate the later form of reasoning to the former. In most approaches, the use of a single clause to express both kinds of knowledge, leads to an asymmetric symbiosis. In this paper we present an extension based on constraint LP (CLP) principles. The use of techniques developed in the clp(X) framework avoids overloading the clausal representation. In particular, we introduce a generic formalism clp(pfd(Y)) which uses finite domains as basic objects on which probability distributions can be attached. Furthermore, the store is extended to hold probabilistic information such as conditional constraints and probabilistic evidence. Uncertain and crisp reasoning is integrated further, by considering predicates containing probabilistic variables as statistical events in an intuitive probability space.

Preliminaries. Our notation follows [5]. Predicates, or atomic formulae, are of the form $r(t, A)$, r is a predicate name and t with A are first-order terms (t a constant and A a logical variable). We extend first-order terms to also include probabilistic variables. Names for logical and probabilistic variables start with a capital letter. A logic program, $\mathcal{P}$, is a set of definite clauses. A substitution θ, e.g. $\{A/a\}$, is a set of variables to term pairs. Applying θ to a formula will replace every occurrence of a variable in θ with its associated term, e.g. $\theta r(t, A)$ is equivalent to $r(t, a)$. Store $\mathcal{S}$ is a set of constraints. We use $P_{\mathcal{S}}(E)$ to denote the probability of E in the context of $\mathcal{P}$ and $\mathcal{S}$. Queries are of the form $?E_1, E_2, \ldots$ where each E_i is a predicate. $?E_1, \ldots$ succeeds iff $\mathcal{P} \cup \mathcal{S} \vdash E_1 \wedge E_2 \wedge \ldots$ ($\vdash$ is the derives operator and $\wedge$ the logical *and*). Each successful derivation often provides a substitution θ, denoting that $\mathcal{P} \cup \mathcal{S} \vdash \theta(E_1 \wedge \ldots)$. We take $\mathcal{P} \cup \mathcal{S} \vdash \neg Q$ to be equivalent to $\mathcal{P} \cup \mathcal{S} \not\vdash Q$. Note that in this case there is no associated θ.

2 Framework and Instances

In clp(pfd(Y)) a probabilistic variable V is introduced to the store with an associated finite domain and a function operating over subsets of this domain. The constraint store can, in addition to clp(fd) constraints, hold probabilistic information about such variables. In a clp(pfd(Y)) store $\mathcal{S}$, probabilistic inference computes $\psi_{\mathcal{S}}(V) = \{(v_1, \pi_1), (v_2, \pi_2), \ldots, (v_n, \pi_n)\}$. Each v_i is an element of V's finite domain and the π_is should define a probability distributions, i.e. $0 \leq \pi_i \leq 1$ and $\sum_i \pi_i = 1$. We let $P_{\mathcal{S}}(V = v_i)$ be the probability attached to element v_i under store $\mathcal{S}$. By definition $P_{\mathcal{S}}(V = v_i) = \pi_i$.

In probability theory an event is a subset of the space of all possible outputs for an experiment. Treating the assignment of all possible values to probabilistic variables as our space of possible outcomes, we can view predicates containing such variables as events in this space. The main intuition is that the probability assigned to events is proportional to the space covered by combinations leading to successful derivations.

Let $pvars(E)$ be the vector of probabilistic variables in predicate E, $\mathcal{P}$ be the program defining E and $\mathcal{S}$ a constraint store. We use $\mathcal{E}_i$ to index the variables in $pvars(E)$. Also let e be a vector collecting one element from the finite domain of each variable in $pvars(E)$ and $P_{\mathcal{S}}(\mathcal{E}_i = e_i)$ the probability attached to value e_i of variable $\mathcal{E}_i$ as defined above. E/e denotes predicate E with its probabilistic variables replaced by their respective elements in e. The probability of predicate E with respect to store $\mathcal{S}$ and program $\mathcal{P}$ is

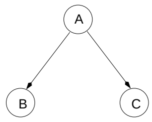
$$P_{\mathcal{S}}(E) = P(E \mid \mathcal{P} \cup \mathcal{S}) = \sum_{\substack{\forall e \\ \mathcal{P} \cup \mathcal{S} \vdash E/e}} P_{\mathcal{S}}(E/e) = \sum_{\substack{\forall e \\ \mathcal{P} \cup \mathcal{S} \vdash E/e}} \prod_i P_{\mathcal{S}}(\mathcal{E}_i = e_i)$$

For example, for program $\mathcal{P}_1$: lucky(iv, hd). lucky(v, hd). lucky(vi, hd). and store $\mathcal{S}_1$ with variables D and C, with $\psi_{\mathcal{S}_1}(D) = \{(i, 1/6), (ii, 1/6), (iii, 1/6), (iv, 1/6), (v, 1/6), (vi, 1/6)\}$ and $\psi_{\mathcal{S}_1}(C) = \{(hd, 1/2), (tl, 1/2)\}$. The probability of a lucky combination is $P_{\mathcal{S}_1}(lucky(D, C)) = 1/4$.

The benefit of regarding predicates containing constraint variables as events is threefold. Firstly the seamless integration of crisp and uncertain reasoning. Secondly the provision of a time point marking prior to posterior distribution transitions. Thirdly the algorithmic separation of crisp and uncertain inference, leading to accountability of resource usage. In comparison to clp(fd) its probabilistic extension provides additional information about the elements in its domain. One important way in which this information can be exploited is during labelling. In clp(pfd(Y)) backtrackable predicate $\mathbf{label}(V, Select, ElPrb, PrbSum)$ can be used to instantiate a probabilistic variable (V) to an element of its finite domain and $ElPrb$ to the element's probability. The order is dictated by $Select$ and $PrbSum$ keeps the sum of the $ElPrb$ encountered so far.

To fully instantiate clp(pfd(Y)) there are three choices to be made: (i) the kind of probabilistic information provided when declaring a probabilistic variable; (ii) the constraints that can be added to the store and (iii) the algorithms for probabilistic inference, which use (i) and (ii) to derive probability distributions for the variables.

clp(pfd(bn)). The framework can be instantiated to a language that introduces Bayesian Networks, BNs [6], reasoning to LP. The importance of such integration has been highlighted in [8] and [3]. The first work embedded BNs to abductive horn clauses. In [3] probability distributions are declared over skolem functions and the emphasis is on machine learning tasks, rather than on probabilistic reasoning. BNs are propositional in that only assignment events ($X = x$ for variable X and value x) are considered. Also, the relationships between variables are widely viewed in a causality context. The basic premise of a BN is that it represents the full probabilistic knowledge of a field. Evidence about particular cases is weighted against the accumulated knowledge. What is sought, is the probabilities of events which explain the evidence. Consider the example BN



	A = y	A = n
B = y	0.80	0.10
B = n	0.20	0.90

	A = y	A = n
C = y	0.60	0.90
C = n	0.40	0.10

Marginal distributions are calculated from the potentials (conditional or joint distribution tables) and the probability of specific cases based on relevant evidence entered in the network. The existence of evidence about C may result to changes in the distribution of B and vice versa. The direction in which BNs use conditional probability is determined by the presence of evidence.

In clp(pfd(bn)) probabilistic variables correspond to nodes in a BN and are introduced with their full conditional probability tables. A variable V with parent nodes the list of variables $Parents$ and conditional table $Table$ is introduced to the store with $cpt(V, Parents, Table)$. In the example BN, variables are declared as $cpt(A, [], [y, n])$, $cpt(B, [A], [(y, y, .8), (y, n, .2), (n, y, .1), (n, n, .9)])$ and $cpt(C, [A], [(y, y, .6), (y, n, .4), (n, y, .9), (n, n, .1)])$

BN algorithms weight currently available evidence, when computing the posterior distributions of variables. In the example, evidence might suggest that the distribution of A is $\{(y, .8), (n, .2)\}$. Using this and $P(C|A)$ we can derive $P(C)$. In this case $P(C) = \{(y, .66), (n, .34)\}$. In clp(pfd(bn)) evidence are stored in the constraint store. The example evidence is added by $evidence(A, [(y, .8), (n, .2)])$. The important factor, in using the store for holding current evidence, is that the introduction of evidence to the store provides a convenient time transition point. The inference algorithms employed in clp(pfd(bn)) are those of probability or belief updating in BNs (see [6]). In BNs the most important predicate is $=$ and inference computes probability of specific values. In our framework this is equivalent to finding the distribution of a probabilistic variable.

clp(pfd(c)). Another instantiation for clp(pfd(Y)) is to clp(pfd(c)) (see [1] for more details). This uses functions over domains to designate basic probabilistic behaviour and conditional constraints for dividing the constraint store into probabilistic fragments. We term this instantiation clp(pfd(c)) in recognition of the important role the conditional constraint plays. A probabilistic variable has two parts: (i) a *finite domain*, which at each stage holds the collection of possible values that can be assigned to the variable and (ii) a *probability assigning function* which is used to assign probabilities to the elements of the domain. The probability function declares the basic statistical behaviour of the variable.

Let Fd be a list of distinct objects representing a finite domain, ϕ_V be a probability function defined over all sublists of Fd and $Args$ a list of ground terms. Probabilistic variables are declared with $V \sim \phi_V(Fd, Args)$. For a subset T of Fd ($T \subseteq Fd$) the result of $\phi_V(T, Args)$ is a list of pairs, each pair coupling an element of T to its associated probability. Each element of T should be given a value and the sum of all given values should be one. $Args$ parameterise aspects of ϕ_V and if it is the empty list we shorten $\phi_V(Set, [])$ to $\phi_V(Set)$. We will use this shorter form when ϕ_V is applied, since $Args$ only play a role in variable declarations. We will also drop the subscript from ϕ_V when the context clearly identifies a particular V. In relation to the ψ notation, the following holds for empty store $\emptyset$: $\phi_V(Fd) = \psi_\emptyset(V)$

For example, $Heat \sim finite\_geometric([l, m, h], 2)$ declares a finite geometric distribution for variable $Heat$. In this case the deterioration factor is 2. The distribution in the absence of other information is $\psi_\emptyset(Heat) = \{(l, 4/7)(m, 2/7), (h, 1/7)\}$. A distinct feature of clp(pfd(c)) is that the two constituents of a probabilistic variable are kept separate. As a result, the variable is still capable of participating in finite domain constraints, thus it is orthogonal to clp(fd) while sharing information, and also probabilistic functions capture statistical behaviour of variables in a manner which is, to a large extent, independent of specific domain values. Adding clp(fd) constraint $Heat \neq m$ to the store, changes the distribution of $Heat$ to $\{(l, 2/3), (h, 1/3)\}$.

The probabilistic information added to the store is the conditional constraint. Its main intuition is that of defining probability subspaces in which we know that certain events hold. Conditional C is of the form: $D_1 : \pi_1 \oplus \ldots \oplus D_m : \pi_m \mid Q$. Each D_i is a predicate and all should share a single probabilistic variable V. Q is a predicate not containing V, and $0 \leq \pi_i \leq 1, \sum_i \pi_i = 1$. V's distribution is altered as a result of C being added to the store, see [1]. The distribution of V according to conditional C, $P_{\{C\}}(V = v)$, is computed by considering all subspaces $(1 \leq j \leq i)$ and by applying ϕ_V within each subspace. clp(pfd(c)) instantiates the proposed framework as follows. Probabilistic information at declaration time is given by ϕ_V, which describes the probabilistic behaviour of the variable after all finite domain pruning. Conditionals add probabilistic information to the store. This information partitions the space to weighted subspaces within which differ-

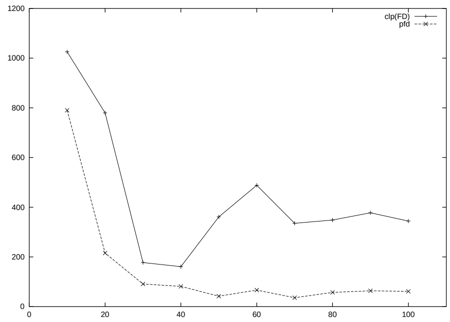


Fig. 1. clp(pfd(c)) versus clp(fd) timings comparison.

ent events hold. Inference uses these partitions and the application of functions to compute updated probability distributions for the conditioned variables.

3 Comparative Example

To illustrate benefits from the additional information in clp(pfd(Y)) when compared to clp(fd) we juxtapose performances of respective programs for a simple Caesar encoding scheme. The two programs are identical bar: (i) distribution over domains in clp(pfd(c)) based on the formula $\frac{|freq(E_i)-freq(D_i)|}{\sum_k |freq(E_i)-freq(D_k)|}$ for E_i an encoded letter, D_i a dictionary letter and $freq()$ their respective frequencies and (ii) labelling in clp(pfd(c)) uses a best-first algorithm. Execution times for ten random sets at points in the range of 10-100 words drawn from a dictionary are shown in Fig. 1. Both programs run on the SICStus engine. One was written in clp(fd) [2] and one in a meta-interpreted implementation of clp(pfd(c)) (see `http://www.cs.york.ac.uk/~nicos/sware/pfds/`).

References

1. N. Angelopoulos. *Probabilistic Finite Domains.* PhD thesis, Department of Computing Science, City University, London, UK, 2001.
2. M. Carlsson, G. Ottosson, and B. Carlson. An open-ended finite domain constraint solver. In *Progr. Languages: Implem., Logics, and Programs*, 1997.
3. V. S. Costa, D. Page, and J. Cussens. CLP($\mathcal{BN}$) constraint logic programming for probabilistic knowledge, 2002. Personal communication, unpublished work.
4. J. Cussens. Stochastic logic programs: Sampling, inference and applications. In *16th Conference on Uncertainty in AI (UAI-2000)*, pages 115–122, 2000.
5. C. J. Hogger. *Essentials of Logic Programming.* OUP, Oxford, 1990.
6. F. V. Jensen. *Bayesian Networks and Decision Graphs.* Springer, 2001.
7. Y. Kameya and T. Sato. Efficient learning with tabulation for parameterized logic programs. In *CL2000*, pages 269–294, July 2000.
8. D. Poole. Probabilistic horn abduction and bayesian networks. *AI*, 64:81–129.
9. S. Riezler. *Probabilistic Constraint Logic Programming.* PhD thesis, Neuphilologische Fakultät, Universität Tübingen, Germany, 1998.