Machine Learning (2020) 109:1371–1392
https://doi.org/10.1007/s10994-019-05842-x

Constructing generative logical models for optimisation problems using domain knowledge

Ashwin Srinivasan[1] · **Lovekesh Vig**[2] · **Gautam Shroff**[2]

Received: 11 February 2019 / Revised: 19 August 2019 / Accepted: 6 September 2019 /
Published online: 13 November 2019

Abstract
In this paper we seek to identify data instances with a low value of some objective (or cost) function. Normally posed as optimisation problems, our interest is in problems that have the following characteristics: (a) optimal, or even near-optimal solutions are very rare; (b) it is expensive to obtain the value of the objective function for large numbers of data instances; and (c) there is domain knowledge in the form of experience, rules-of-thumb, constraints and the like, which is difficult to translate into the usual constraints for numerical optimisation procedures. Here we investigate the use of Inductive Logic Programming (ILP) to construct models within a procedure that progressively attempts to increase the number of near-optimal solutions. Using ILP in this manner requires a change in focus from discriminatory models (the usual staple for ILP) to generative models. Using controlled datasets, we investigate the use of probability-sampling of solutions based on the estimated cost of clauses found using ILP. Specifically, we compare the results obtained against: (a) simple random sampling; and (b) generative deep network models that use a low-level encoding and automatically construct higher-level features. Our results suggest: (1) Against each of the alternatives, probability-sampling from ILP-constructed models contain more near-optimal solutions; (2) The key to the effectiveness of ILP-constructed models is the availability of domain knowledge. We also demonstrate the use of ILP in this manner on two real-world problems from the area of drug-design (predicting solubility and binding affinity), using domain knowledge of chemical ring structures and functional groups. Taken together, our results suggest that generative modelling using ILP can be very effective for optimisation problems where: (a) the number of training instances to be used is restricted, and (b) there is domain knowledge relevant to low-cost solutions.

Keywords Domain knowledge guided optimisation · Inductive logic programming · Generative models

Editors: Dimitar Kazakov and Filip Zelezny.

✉ Ashwin Srinivasan
ashwin@goa.bits-pilani.ac.in

Extended author information available on the last page of the article

1 Introduction

Large planning problems are often tackled using heuristic search that exploits domain knowledge, rather than via direct numerical optimisation, which is often intractable. Consider, for example, nationwide railway scheduling. This involves generating a feasible and safe schedule for the entire railway operations involving trains and railway stations in the network. Optimal railway scheduling seeks to find an assignment of trains to track segments (or 'blocks') so as to minimize or eliminate delays, or, more generally minimise costs, while ensuring that that every prescribed train has a contiguous, conflict-free path to its destination. The railway scheduling problem can be modelled either as a specific form of the general job-shop scheduling problem (JSP) with blocking and no-wait constraints (D'Ariano et al. 2007), or as a binary integer optimisation problem (Harrod 2010). In the latter, the finite occupancy of a segment of track by a single train for a discrete time duration is encoded using a binary variable $x^r_{i,t}$, where r indicates one of R trains, t is one of T time slots, and i refers to one of B^r track segments that compose one or more feasible paths for train r. The scheduling objective is then to minimise the total cost, i.e., min $\sum c^r_{i,t} x^r_{i,t}$, where $c^r_{i,t}$ is a cost associated with the assignment $x^r_{i,t}$, incorporating elements such as delay etc., subject to a number of flow constraints such as $\sum_r x^r_{i,t} = 1,\ \forall r \in R$. The complete set of constraints, which include other single-commodity flow constraints such as those above, as well as other multi-commodity constraints, together ensure the resolution of spatio-temporal contentions for track segments by ordering the trains on a line section or on a loop at a station, i.e., by determining the time instances at which each train enters and exits (arrival and departure) each line section and loop on its route.

Whichever formulation is used, the problem of finding an optimal schedule is NP-hard, and even for a small instance it is usually a challenge to get a reasonable solution that can then be tweaked towards optimality. For large-scale problems, it is practically impossible to guarantee optimality: India, for example, has about 8000 stations and runs 12,000 trains a day. By conservative estimates, this would result in about 2 million binary variables and 3.5 million real-valued variables for the optimisation formulation just described. So what is usually done? In practice, schedules are generated using knowledge that is qualitative, and not easily encoded in a form suitable for numerical optimisation. Here, for instance, are some guiding principles that are followed by the Australian Rail Track Corporation when scheduling trains: (1) If a "healthy" train is running late, it should be given equal preference to other healthy trains (there is a definition of healthy in the guidelines, which is not important here); (2) A higher priority train should be given preference to a lower priority train, provided the delay to the lower priority train is kept to a minimum; and so on. It is evident from this that train-scheduling may benefit from knowing if a train is healthy, what a train's priority is, and so on. But are priorities and train-health fixed, irrespective of the context? What values constitute acceptable delays to a low-priority train? Generating good train-schedules will require a combination of quantitative knowledge of a train's running times and qualitative knowledge about the train in isolation, and in relation to other trains.

In this paper, we propose a heuristic search method, that comes under the broad category of an estimation distribution algorithm (EDA: Pelikan et al. 2000). EDAs iteratively generate better solutions to an optimisation problem using machine-constructed models. Usually, these models have been generative probabilistic models, like Bayesian Networks. Domain knowledge has then to be translated into a specification of the prior distribution for such networks (often, the choice of prior distribution is restricted to allow efficient parameter estimation), and perhaps some constraints on the topology of the Bayesian) network. Here we are concerned with problems for which a translation into prior distribution functions over parameter values or network structure, is not evident [although see Angelopoulos and Cussens (2008) for flexible Bayesian priors]. Our interest in ILP derives from the approach presenting an extremely flexible way to use domain knowledge when constructing models. In experiments reported here, we do not take on a problem as ambitious as nationwide train-scheduling. Instead, we focus on two controlled (but non-trivial) synthetic problems and two uncontrolled real-world problems from the area of drug-design.

The principal contributions of this paper are as follows:

1. To the specific area of ILP, we provide an application of constructing models not for discriminating amongst instances (as is done usually), but to generate new instances; and
2. To the broader area of machine learning we provide evidence of the utility of using a technique that is capable of including domain knowledge for model-based identification of good solutions to optimisation problems.

The rest of the paper is organised as follows. Sect. 2 provides a brief description of the heuristic search we propose to employ for optimisation problems. Section 2.1 describes how ILP is used within the iterative loop of the search. The use of ILP in this manner requires: (a) models it constructs to be generative; and (b) a sampling procedure. These requirements are addressed in Sect. 2.2. Section 3 describes an empirical evaluation followed by conclusions in Sect. 4.

2 Evolutionary search for near-optimal solutions

The basic search method we use is inspired by the MIMIC algorithm (De Bonet et al. 1997), which uses an evolutionary procedure for model-assisted sampling. Assuming that we are looking to minimise an objective function $F(\mathbf{x})$, where $\mathbf{x}$ is an instance from some instance-space $\mathcal{X}$, the approach first constructs an appropriate discriminatory model to distinguish between samples of values below and above some thresold θ, i.e., $F(\mathbf{x}) \leq \theta$ and $F(\mathbf{x}) > \theta$. The model is then used to generate the population for the next iteration, while also lowering θ. A generic procedure is in Fig. 1.

Clearly, the procedure in Fig. 1 requires some kind of generative model which is used to generate new instances (Step 2c). Usually, the prefered choice is a probabilistic model, like a Bayesian Network. However sampling from such networks can be notoriously inefficient, especially if the number of variables are large. In addition, domain knowledge may not be of the kind that can translate easily into statements about conditional dependencies, or priors over network structures. For these reasons, interest is increasing in the use of deep

```
Procedure EOMS: Evolutionary Optimisation using Model-Assisted Sampling
    1. Initialise population $P := \{\mathbf{x}_i\}$; $\theta := \theta_0$
    2. while not converged do
        (a) for all $\mathbf{x}_i$ in $P$ $label(\mathbf{x}_i) := 1$ if $F(\mathbf{x}_i) \leq \theta$ else $label(\mathbf{x}_i) := 0$
        (b) train model $M$ to discriminate between 1 and 0 labels (for example, $M$ classifies
            $\mathbf{x}$ as 1 if $Pr(label(\mathbf{x}) = 1|M, \mathbf{x}) > Pr(label(\mathbf{x}) = 0|M, \mathbf{x})$)
        (c) regenerate $P$ by repeated sampling using model $M$
        (d) reduce threshold $\theta$
    3. return $P$
```

Fig. 1 Evolutionary optimisation using machine learning models to guide sampling. Here the term "model" is used in a data-analytic sense, of being a description of data, rather than in the logical sense of being an interpretation that makes a formula true

generative network models, that have efficient sampling procedures, and are able to construct automatically relevant domain-specific features from low-level representations. We look at some options before we consider the use of generative ILP models. Widely used examples of deep generative networks are Deep Belief Nets (DBNs), composed of multiple latent variable models called Restricted Boltzman Machines (RBMs). RBMs belong to the class of Energy Based Models (EBMs) and can be interpreted as parameterized probabilistic graphical models (Fischer and Igel 2012).

Another kind of deep generative model that has gained significant prominence is a generative adversarial network (or GAN: Goodfellow et al. 2014). GANs are unique amongst deep generative models in that they do not employ a user-defined cost function for the generator. GANs are trained using two competing networks, a dicriminator and a generator. In this paper we use a variant of GANs called conditional adversarial networks (cGANs), which include a set of conditional variables Y as an additional input layer for the discriminator and generator. For our problem there is a single conditional variable (denoting the “1” or “0” class values in the EOMS procedure).

From now on, we will use the term “EODN” (Evolutionary Optimisation using Deep Networks) to refer to the EOMS procedure in which the M used is a deep generative model (specifically, either a DBN or a GAN). We contrast this with an EOMS procedure with a model obtained using Inductive Logic Programming (ILP).

2.1 ILP-assisted evolutionary optimisation

We now look at the use of ILP as the basis for the model M in the EOMS procedure. The principal motivation for the use of ILP has already been described earlier: ILP provides an extremely flexible way to construct models using domain knowledge. While prominent applications of ILP (for example, the line of work on structure-activity relations by King et al. (1995) and Srinivasan et al. (1997) and early specifications of ILP [for example, in Muggleton and Raedt (1994)] have largely been concerned with discriminatory models, this does not necessarily have to be case. Research in probabilistic inductive logic programming (De Raedt et al. 2008), and earlier work on stochastic logic programming (Muggleton 1996), do allow the use of theories in a generative manner. The use of ILP in this section can be seen as a simple form of probabilistic ILP, in which we deal separately with model-construction and inference for sampling (with the probabilistic aspect only concerned being concerned with the latter: more on this in a later section). The procedure EOIS in Fig. 2 is a refinement of the EOMS procedure above, using ILP as the vehicle for model-construction.

Assume we are provided with domain knowledge encoded as a set of definite clauses B, and examples E consisting of instances $\mathbf{x}$ s.t. $F(\mathbf{x}) \leq \theta_k$ for some threshold θ_k (positive instances, or E^+) and instances for which $F(\mathbf{x}) > \theta_k$ (negative instances, or E^-) respectively. Then, in EOIS, $ilp(B, E^+, E^-)$ is an ILP algorithm that returns a set of definite clauses M s.t. $B \cup M \models E^+$; and $B \cup M \cup E^- \not\models \Box$. Given instances $\mathbf{x}$ are drawn from a population P, assume the ILP algorithm has found some model M ($M \neq \emptyset$). Then the function $sample(P, n, M, B.E)$ returns a set S of at most n instances from $P \backslash E$ s.t. for each $\mathbf{x} \in S$, $B \cup M \models label(\mathbf{x}, pos)$. Practically speaking, we derive (infer) $F(\mathbf{x}) \leq \theta_k$ from $B \cup M$. Note: the actual F value of $\mathbf{x}$ may be above or below the corresponding value θ_k. If $M = \emptyset$, $sample$ returns a random selection of n instances from $P \backslash E$. We note on any iteration $k > 0$, $E_{k-1} \subseteq E_k$. Also since $\theta_k < \theta_{k-1}$, $E^-_{k-1} \subseteq E^-_k$. The procedure terminates simply because the sequence of θ_k's is finite.

```
Procedure EOIS: Evolutionary Optimisation using ILP-Assisted Sampling

Given: (a) A population $P$ of instances; (b) Background knowledge $B$; (c) an upper bound
    $\theta^*$ on the cost of acceptable solutions; (d) a finite decreasing sequence of cost values
    $\theta_1, \theta_2, \ldots, \theta_n$ s.t. $\theta_1 > \theta_2 > \cdots > \theta_n$; and (e) an upper bound on the sample size $n$

    1. Let $M_0 := \emptyset$ and $E_0 = \emptyset$
    2. Let $P_0 := sample(P, n, M_0, B, E_0)$
    3. Let $k = 1$
    4. while $(\theta_k \geq \theta^*)$ do
       (a) Let $E_k = P_{k-1} \cup E_{k-1}$
       (b) ${E_k}^+ := \{label(\mathbf{x}_i, pos) : \mathbf{x}_i \in E_k$ and $F(\mathbf{x}_i) \leq \theta_k\}$ and ${E_k}^- := \{\neg label(\mathbf{x}_i, pos) :$
           $\mathbf{x}_i \in E_k$ and $F(\mathbf{x}_i) > \theta_k\}$
       (c) $M_k := ilp(B, {E_k}^+, {E_k}^-))$
       (d) $P_k := sample(P, n, M_k, B, E_k)$
       (e) increment $k$
    5. return $P_{k-1}$
```

Fig. 2 Evolutionary optimisation using ILP models to guide sampling. For simplicity, we do not distinguish between an instance $\mathbf{x} \in P$, and its logical encoding in a form suitable for an ILP engine

In general therefore, we require *sample* to draw instances from the success-set of the ILP-constructed theory M, since these represent the good solutions. We describe this next.

2.2 Sampling using an ILP model

Let us assume that on any iteration k of EOIS the ILP model M_k is a logic program consisting of definite clauses for the target predicate $label/2$ as the positive literal (that is, the head of the clause). On iteration k, we require to be able to generate a sample from the success-set of M_k.[1] We will look first at the inference procedure.

It is the intent of sampling to generate low-cost instances (or at least to generate instances expected to have low cost). It is well-known that the SLD-resolution procedure used by Prolog requires the specification of a computation-rule ("which literal?") and a search-rule ("which clause?"). Standard Prolog uses a leftmost literal computation rule, and a search rule that amounts to selecting clauses in order of appearance in a program. Of these, for non-recursive clauses of the kind we will be considering here, the choice of the search rule is of special importance to determining elements of the success-set (Lloyd 1987).[2] It is of interest here to consider a search rule that is dependent on the average costs of instances entailed by the clauses in M_k

Recall that clauses in M_k are constructed by an ILP engine, given $\mathbf{x}$'s with F-values above and below some threshold value θ_k. On each iteration k, with training set $E_k = E_k^+ \cup E_k^-$ and any clause $C \in M_k$, we define the instances covered by C, given background knowledge B as follows:

$$Covers_B(C) = \{\mathbf{x} : label(\mathbf{x}, pos) \in E_k^+ \text{ and } B \wedge C \models label(\mathbf{x}, pos)\}$$

[1] Correctly, we mean the least Herbrand model of M_k. However, we have elected to use the procedural notion of the success-set to avoid confusion with the data-theoretic use of the term model which we have employed. Moreover, we will be generating samples using resolution-based inference, so the procedural notion is the one that matters in practice.

[2] In general, given a set of clauses $\mathcal{C}$, and a partial ordering $\succeq$ defined over elements of $\mathcal{C}$, a search rule is a total ordering $\succ$ consistent with $\succeq$. Variations arise with the definition of $\succeq$ and the subsequent total ordering selected.

$$\cup\{\mathbf{x} : \neg label(\mathbf{x}, pos) \in E_k^- \text{ and } B \wedge C \models label(\mathbf{x}, pos)\}$$

We will usually call this $Covers(C)$. The costs of instances covered by C is:

$$Costs(C) = \{f : \mathbf{x} \in Covers(C) \text{ and } f = F(\mathbf{x})\}$$

and we are able to define $AvCost(C)$ as the mean of the values in $Costs(C)$[3]:

$$AvCost(C) = \text{Mean}(Costs(C))$$

Then, we propose using a cost-sensitive search rule in which clauses with lower mean cost have a higher probability of being selected by the inference procedure. Using SLD inference with this kind of probabilistic selection is equivalent to backtrackable sampling from a stochastic logic program (Cussens 2000).

We turn now to the question of how the model M_k can be made generative. We know clauses in M_k will be of the form (shown here in a kind of pseudo-Prolog format):

$$\begin{array}{ll} label(X, pos) \leftarrow & \\ & \text{(Body literals)} \end{array}$$

where the body literals use predicates defined in the background knowledge. In logic-programming, such a clause being "generative" simply means that we are able to obtain answer-substitutions for X. But there is a small twist: we would like the answer-substitutions to be grounding substitutions from the instance-space P. Assuming that instances of P belong to some type T and that elements of T can be enumerated, then either one of the rewrites below of the clause will ensure substititutions are of the kind we seek:

$$\begin{array}{ll} label(X, pos) \leftarrow & \\ & type_T(X), \\ & \text{(Body literals)} \end{array} \qquad \begin{array}{ll} label(X, pos) \leftarrow & \\ & \text{(Body literals)} \\ & type_T(X). \end{array}$$

where $type_T$ is a meta-predicate that is true for any instance x of type T. It is used here as an enumerator of elements of T (left), or to ensure answer substitutions x for X are elements of T (right). If the background predicates are ground atoms, then the rewrite on the right is more efficient. In either case, we will call the rewritten clause as the "generative version of the clause".[4]

The main steps of the procedure *sample* used in EOIS procedure are in Fig. 3.

2.3 Note on dominance information

A special form of domain knowledge in optimisation problems is that of *dominance*. We follow (Jouglet and Carlier 2011), in which a dominance rule specifies a subset of the solution space that contains some optimal (or in our case, near-optimal) solutions (see Fig. 4).

Operationally, dominance constraints are used to reduce the search-space. ILP systems like Srinivasan (1999) have two natural ways of incorporating dominance information. First, clauses examined by the ILP system for inclusion in a model can be *pruned*, if we are sure that they will generate solutions outside the dominant subset:

[3] Correctly, what we are doing is defining an aggregation function $A : \Re_C^d \rightarrow \Re$ where for any $C \in M_k$, $d_C = |Costs(C)|$. The mean is a special case of such an aggregation function.

[4] In practice, we will use the type-definitions provided to construct the generative version. In effect, we have rewritten the clause into what has been called in the ILP-literature as a *range-restricted* form. A definite clause is range-restricted if every term that appears in the consequent (literals to the left of the $\leftarrow$) also appears in the antecedent (literals to the right of the $\leftarrow$).

```
Procedure sample: Generate a sample of instances using an ILP model

Given: (a) A population of instance $P$; (b) An upper bound on the number of instances $n$;
    (c) A logic program $M$ (the ILP model); (d) Background knowledge $B$; and (e) A set of
    previously sampled (training) instances $E$

    1. If $M = \emptyset$ return $Uniform(n, P)$
    2. Let $Gen := \{C^* : C \in M$ and $C^*$ is the generative version of $C\}$
    3. Let $Costs := \{(f_{C^*}, C^*) : C^* \in Gen$ and $f_{C^*} = AvCost(C^*)\}$
    4. Let $Goal := (\leftarrow label(X, pos))$
    5. Let $S := \emptyset$ and $Done := false$
    6. repeat
        (a) Let $Costs^* = reorder(Costs)$
        (b) Let $M^*$ be the sequence of clauses in $Costs^*$
        (c) Let $Program$ be the sequence of clauses in $B$ concatenated with the sequence of
            clauses $M^*$
        (d) If $X/\mathbf{x} \notin sld(Program, Goal)$ then $Done := true$
        (e) Otherwise if $label(\mathbf{x}, pos) \notin E$ then $S := S \cup \{\mathbf{x}\}$
        (f) If $|S| = n$ then $Done := true$
    7. until Done
    8. return $S$
```

Fig. 3 Generating samples using the ILP model. Here, *Uniform* returns a uniform random sample of size at most n from the P, and *reorder* is non-deterministic, returning a total ordering (i.e. a sequence) of elements (f_i, c_i) in *Costs*, generated by using probabilistic sampling based on the f_i values of tuples in *Costs*). $sld(\cdot)$ returns set of answer substitution from refutations of *Goal* using Program. If *reorder* is deterministic (for example, a sequence sorted on f values), then M^* need be computed only once, and can be obtained Step 6. The loop in Steps 6–7 terminates when $|S| = n$ or no answer-substitutions are possible from $sld(\cdot)$)

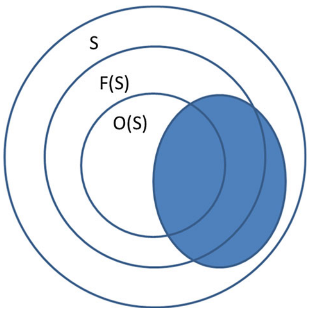


Fig. 4 Dominant solutions in optimisation (adapted from Jouglet and Carlier 2011). Here S denotes a set of possible solutions, $F(S)$ denotes feasible solutions, $O(S)$ the subset of (near-)optimal solutions. Dominance constraints are used to specify the shaded subset ($\delta(S)$). Solutions in $\delta(S)$ are said to dominate solutions in $S \backslash \delta(S)$. Ideally we would like $O(S) \subseteq \delta(S)$

$$prune(C) \leftarrow \\ \text{(Clause } C \text{ fails a dominance constraint)}$$

Alternatively, we can define a *refinement operator* that enumerates elements of the search space, by actively incorporating dominance criteria:

$$refine(C_i, C_j) \leftarrow \\ \text{(Clause } C_j \text{ satisfies dominance constraints)}$$

Here the ILP system will start the search from some known clause C_0 and repeatedly call the *refine* predicate to obtain an enumeration of clauses $C_1, C_2, \ldots$, all of which will satisfy some set of necessary dominance constraints. In either case, the clauses in the resulting ILP model will satisfy the dominance constraints encoded as domain knowledge, and we can expect the subsequent generative model to be more constrained than one obtained without dominance criteria. In experiments that follow, we do not employ such constraints: suffice to note that such constraints can be employed by an ILP system, if available.

We are now in a position to clarify further what we mean by the claim that the use of domain knowledge by an ILP system can lead to more effective models for identifying near-optimal instances. In this paper, this will mean that the EOIS procedure, equipped with an ILP engine and relevant background knowledge, will result in generating more near-optimal instances. Below, we investigate this empirically.

3 Empirical evaluation

3.1 Aims

The principal aim of the empirical evaluation is to investigate the performance of the EOIS procedure, given relevant domain knowledge, against: (a) simple random sampling from the instance space which uses no domain knowledge; and (b) evolutionary optimisation that uses methods that construct generative models by automatically constructing relevant domain-level features from low-level data.
We use evolutionary optimisation using deep generative networks (EODNs) as representative of (b). In the first instance, we will assess performance of EOIS as follows: at the end of each iteration of the evolutionary optimisation procedure, we compute the number of near-optimal instances identified and compare against alternatives.[5] Recall that by "near-optimal instances" we mean those instances for which the cost is below some pre-specified value $\theta*$.

A question that follows naturally is this: what if the domain knowledge available for EOIS is irrelevant, or of limited relevance? One of the two synthetic datasets we have chosen is of this kind, and we will investigate what can be expected in such circumstances.[6]

3.2 Materials

3.2.1 Synthetic data

We use two synthetic datasets, one arising from the KRK chess endgame (an endgame with just White King, White Rook and Black King on the board), and the other a restricted, but nevertheless hard 5×5 job-shop scheduling (scheduling 5 jobs taking varying lengths of time onto 5 machines, each capable of processing just one task at a time).

The optimisation problem we examine for the KRK endgame is to predict the depth-of-win with optimal play (Bain and Muggleton 1994). Although this problem has not been as popular in ILP as the task of predicting "White-to-move position is illegal" (Bain 1991; Muggleton et al. 1992), it offers a number of advantages as a synthetic test-bed for problems of interest to us. First, as with other chess endgames, KRK-win is a complex, enumerable

[5] We do not use a formal definition of relevance in this paper, but use the term here in an operational sense. That is, domain knowledge is relevant if it allows consistent descriptions of low-cost solutions within the representation language. In practice, this requires a sufficiently powerful ILP engine that can find such descriptions, if they exist. A recent characterisation of relevance in terms of generality of predicates has been proposed by Patzantsis and Muggleton; and earlier work has characterised it in terms of optimal answers to linear cost functions (Srinivasan 2001). It is of interest, but outside the scope of this paper, to formalise the notion of relevance here in terms of either of these approaches.

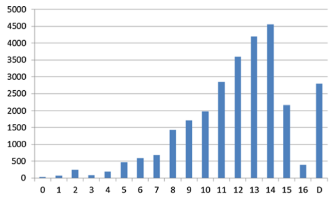
[6] For the ILP practitioner, matters are not always as cut-and-dried, and the question of interest is more akin to: what if it is not known how relevant the background knowledge is for the problem? The results from the investigation here then provide some clues on the relevance or otherwise of the background knowledge.

Cost	Instances	Cost	Instances
0	27 (0.001)	9	1712 (0.196)
1	78 (0.004)	10	1985 (0.267)
2	246 (0.012)	11	2854 (0.368)
3	81 (0.152)	12	3597 (0.497)
4	198 (0.022)	13	4194 (0.646)
5	471 (0.039)	14	4553 (0.808)
6	592 (0.060)	15	2166 (0.886)
7	683 (0.084)	16	390 (0.899)
8	1433 (0.136)	draw	2796 (1.0)

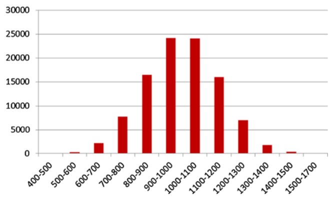
Total Instances: 28056

Cost	Instances	Cost	Instances
400–500	10 (0.0001)	1000–1100	24067 (0.748)
500–600	294 (0.003)	1100–1200	15913 (0.907)
600–700	2186 (0.025)	1200–1300	7025 (0.978)
700–800	7744 (0.102)	1300–1400	1818 (0.996)
800–900	16398 (0.266)	1400–1500	345 (0.999)
900–1000	24135 (0.508)	1500–1700	66 (1.0)

Total Instances: 100000



(a) Chess



(b) Job-Shop

Fig. 5 Distribution of cost values. Numbers in parentheses in the tabulation are cumulative proportions

domain for which there is complete, noise-free data. Second, optimal “costs” are known for all data instances. Third, the problem has been studied by chess-experts at least since Torres y Quevado built a machine, in 1910, capable of playing the KRK endgame. This has resulted in a substantial amount of domain-specific knowledge. We direct the reader to Breda (2006) for the history of automated methods for the KRK-endgame. For us, it suffices to treat the problem as a form of optimisation, with the cost being the depth-of-win with Black-to-move, assuming minimax-optimal play. For us, the availability of significant amounts of domain knowledge for near-optimal play makes this what we term a “knowledge-rich” problem. In principle, there are $64^3 \approx 260{,}000$ possible positions for the KRK endgame, not all legal. Removing illegal positions, and redundancies arising from symmetries of the board reduces the size of the instance space to about 28,000 and the distribution shown in Fig. 5a. The sampling task here is to generate instances with depth-of-win equal to 0. Simple random sampling has a probability of about 1/1000 of generating such an instance once redundancies are removed.

The job-shop scheduling problem is less controlled than the chess endgame, but is nevertheless representative of many real-life applications (like scheduling trains), which are, in general, known to be computationally hard. We use a job-shop problem with five jobs, each consisting of five tasks that need to be executed in order. These 25 tasks are to be performed using 5 machines, each capable of performing a particular task for any of the jobs. For simplicity, we assume each machine is specialised for a task, a 5×5 matrix defines how long task j of job i takes to execute on machine j. The domain knowledge for this problem is not as well-developed as for the chess problem, and it is not clear how much of it is relevant to low-cost solutions. For us, the job-shop problem here will constitute an example of a “knowledge-deficient” problem.

Data instances for Chess are in the form of 6-tuples, representing the rank and file (X and Y values) of the 3 pieces involved. Data instances for Job-Shop are in the form of schedules defining the sequence in which tasks of different jobs are performed on each machine, along with the total cost (i.e., time duration) implied by the schedule. A tabulation of numbers of instances and their costs is in Fig. 5.

3.2.2 Real-world data

Besides performance on synthetic data, it is clearly helpful to know what can be expected on real-world problems. We use the following datasets:

Dataset	Task	Total instances	Near-optimal instances
ADME	Drug solubility	≈ 1300	≈ 85
Malaria	Drug efficacy	$\approx 13{,}000$	≈ 170

"ADME" refers to the dataset in Hou et al. (2004), to predict solubility of drug-like molecules. Specifically, the predictions are for log$S$ where S is the solubility in mol/l of the molecule in water at a pH of 7.4. Data are available in the form of the 2-d structure of the molecules (the atom and bond structure). Some additional bulk properties can be obtained or estimated from this structure: we obtain the molecular weight, which is a straightforward computation. We consider the optimisation problem to be one of identifying soluble molecules.

"Malaria" refers to The Tres Cantos Antimalarial (TCAMS) dataset. It is available at the ChEMBL Neglected Tropical Disease archive (www.ebi.ac.uk/chemblntd). The TCAMS database is a result of screening GlaxoSmithKline's library of approximately 2 million compounds. The database consists of 13,000 of the chemicals that were found, on screening, to inhibit significantly the growth of the 3D7 strain of *P. falciparum* in human erythrocytes (Gamo et al. 2010). Data made available include some bulk-properties of the compounds (like molecular weight and hydrophobicity). We will consider the task of identifying molecules with high inhibition activity (inhibition activity ranges from approximately 5.6–8.6: we are attempting to identify molecules with activity 7.0 and above).
Without access to specialised expertise, we are unable to characterise the real-world problems as knoweldge-rich or knowledge-deficient.

3.2.3 Domain knowledge

For Chess, background predicates encode the following (WK denotes the White King, WR the White Rook, and BK the Black King): (a) Distance between pieces WK-BK, WK-BK, WK-WR; (b) File and distance patterns: WR-BK, WK-WR, WK-BK; (c) "Alignment distance": WR-BK; (d) Adjacency patterns: WK-WR, WK-BK, WR-BK; (e) "Between" patterns: WR between WK and BK, WK between WR and BK, BK between WK and WR; (f) Distance to closest edge: BK; (g) Distance to closest corner: BK; (h) Distance to centre: WK; and (i) Inter-piece patterns: Kings in opposition, Kings almost-in-opposition, L-shaped pattern. We direct the reader to Breda (2006) for the history of using these concepts, and their definitions.

For Job-Shop, background predicates encode: (a) schedule job J "early" on machine M (early means first or second); (b) schedule job J "late" on machine M (late means last or second-last); (c) job J has the fastest task for machine M; (d) job J has the slowest task for machine M; (e) job J has a fast task for machine M (fast means the fastest or second-fastest); (f) Job J has a slow task for machine M (slow means slowest or second-slowest); (g) Waiting time for machine M; (h) Total waiting time; (i) Time taken before executing a task on a machine. Correctly, the predicates for (g)–(i) encode upper and lower bounds on times, using the standard inequality predicates $\leq$ and $\geq$.

For both the real-world tasks, domain knowledge is in the form of general chemical knowledge of ring-structures and some functional groups. Background-knowledge contains definitions used for concepts such as: alcohols, aldehydes, halides, amides, amines, acids. esters, ethers, imines, ketones, nitro groups, hydrogen donors and acceptors, hydrophobic groups, positive- and negatively-charged groups, aromatic rings and non-aromatic rings, hetero-rings, 5- and 6-carbon rings and so on. These have been used in structure-activity applications of ILP before (King et al. 1995; Srinivasan et al. 1997). We note that none of these definitions are specifically designed for the tasks of predicting solubility or inhibiting malarial targets.In addition, there are two bulk properties of the molecules that are available: the molecular weight, and partition coefficient values ($\log P$).

3.2.4 Algorithms and machines

The ILP-engine we use is Aleph (1999: we use Version 6, available from A.S. on request). All ILP theories were constructed on an Intel Core i7 laptop computer, using VMware virtual machine running Fedora 13, with an allocation of 2 GB for the virtual machine. The Prolog compiler used was Yap, version 6.1.3.[7]

3.3 Method

Our method is straightforward:

For each optimisation problem, and domain knowledge B:

Using a sequence of threshold values $\langle\theta_1, \theta_2, \ldots, \theta_n\rangle$, with $\theta^* = \theta_n$. On iteration k $(1 \leq k \leq n)$:

1. Obtain an estimate of the number of instances with $F(\mathbf{x}) \leq \theta_n$) using a simple random sample (SRS), EOIS and EODN from the instance space (the EOIS and EODN models are obtained for discriminating between $F(\mathbf{x}) \leq \theta_k$ and $F(\mathbf{x}) > \theta_k$)
2. Compare the numbers of near-optimal estimates obtained by the methods.

We clarify some additional details concerning the method above.

3.3.1 Thresholding

1. The sequence of thresholds for Chess are $\langle 8, 4, 0\rangle$. For Job-Shop, this sequence is $\langle 1000, 750, 600\rangle$; Thus, $\theta^* = 0$ for Chess and 600 for Job-Shop, which means we require exactly optimal solutions for Chess. For the real-world datasets, the thresholds are $\langle -4.0, -2.0$, and $-1.0\rangle$ for ADME; and $\langle 6.5, 7.0, 7.5\rangle$ for Malaria.
2. There is no specific theory underlying the choice of the threshold sequence (although a greedy-search for identifying the sequence is discussed later) , or that of θ^*. For the latter, the choice of 0 for Chess has a problem-specific meaning (checkmate). Using Chess as the baseline, we have chosen θ^* values roughly by increasing the proportion of good solutions by an order of magnitude for Job-Shop and at 1% for the real-world

7 http://www.dcc.fc.up.pt/~vsc/Yap/.

problems (that is, good drugs are in the top 1-percentile of activity). The evolutionary procedures EOIS or EODN only require the sequence θ_i decrease to a value below θ^*, without specifying how this is to be done. It is possible that an adaptive procedure could be devised (more on this later).

3.3.2 Deep network parameter tuning

For training both the DBN and the GAN in every iteration, we utilized a grid search over the number of hidden layers(limited to two due to the relatively small input dimensions), the number of units in each layer (30, 50, 70, 90), and the learning rate (0.0001, 0.001, 0.01). In each iteration the number of novel low cost samples generated was taken to be the tuning criteria. The networks were implemented on a 64 GB server with a 16 GB Volta GPU.

1. The DBN hyper-parameters are tuned by observing the pseudo log-likelihood during training. During training an additional feature bit was concatenated to the data to indicate whether the sample was below (1) or above(0) the current threshold. Sampling was performed in the usual fashion via Gibbs sampling in the topmost layer of the DBN, with a conditional feature bit clamped to 1 during sampling to bias the sampling towards samples lying below the current threshold.
2. The conditional GAN was trained similarly with a feature bit set to 1 during training for samples below the current threshold, and 0 otherwise. The discriminator loss was used to tune the hyper-parameters of the GAN model, with models yielding higher loss being favoured. During generation the conditional feature bit was set to 1 to bias the sampling towards lower cost samples.
3. For both the DBN and the GAN, the number of samples below the current threshold value were over-sampled during training to ensure they constituted 50% of the training data. This ensured that good samples were adequately represented during training. 1000 samples were generated at each iteration and their costs evaluated.
4. Both the discriminator and generator in the GAN were trained as feedforward networks, with the output layer of the generator consisting of sigmoid units and the hidden layers comprising of rectified linear units. The input to the generator is a 10 dimensional gaussian noise vector and a conditional bit to bias it towards low cost solutions.
5. Each successive layer in the DBN was trained via contrastive divergence, with the bottom most data layer initialized to a random data vector, followed by repeated sampling of visible and hidden units for fixed number of iterations (we chose 3 as it yielded the best sampling results).

3.3.3 ILP parameter settings

1. Experience with the use of ILP engine used here (Aleph) suggests that the most sensitive parameter is the one defining a lower-bound on the precision of acceptable clauses (the $minacc$ setting in Aleph). We report experimental results obtained with $minacc = 0.7$ for all datasets, which has been used in previous experiments with ILP. The domain knowledge for Job- Shop does not appear to be sufficiently powerful to allow the identification of good theories with short clauses. That is, the usual Aleph setting of up to 4 literals per clause leaves most of the training data ungeneralised. We therefore allow an upper-bound of up to 10 literals for Job-Shop, with a corresponding increase in the number of search nodes to 10,000 (Chess uses the default setting of 4 and 5000 for these parameters).

2. In the EOIS procedure, the bound on sample size n is 1000 for all problems except ADME, which is a small dataset. For ADME n is 250. The initial sample is obtained using a uniform distribution over all instances. Let us call this P_0. On the first iteration of EOIS ($k = 1$), the datasets ${E_1}^+$ and ${E_1}^-$ are obtained by computing the (actual) costs for instances in P_0, and an ILP model $M_{1,B}$ is constructed.
3. The sample procedure requires a cost-sensitive probability distribution. We use a simple negative exponential function $Ae^{-\alpha c}$ where c is the cost. For all problems, we take $A = \alpha = 1$. The resulting values are normalised to sum to 1, and form the probability distribution from which clauses are selected before generating an instance.
4. There are two sources of sampling variation in EOIS: the initial sample, drawn uniformly from the instance space; and variation arising from the probabilistic selection rule used by the inference engine. Of these, we account for the latter, by obtaining estimates of near-optimal estimates from 10 repetitions of sampling. Variation due to the initial sample affect all methods equally.
5. In all cases, a generative version of the ILP model is obtained by including a type enumerator in all clauses. It is expected that such an enumerator is defined as part of the domain knowledge provided to the ILP engine. This allows us to use standard Prolog inference to generate answer-substitutions.

3.4 Results

Results on the synthetic data are in Table 1. The principal finding in the tabulations are these:

(1) For both problems, the numbers of near-optimal instances with ILP-assisted models is higher than with other models;
(2) Despite identifying a higher number of near-optimal instances on both problems, there is a difference in the performance of EOIS. Specifically, the "performance-gap" in Chess is much greater than on Job-Shop, and the standard deviations suggest the differences may not be statistically significant; and
(3) All results have been obtained by sampling a small portion of the instance space (about 10% for Chess, and about 3% for Job-Shop).

We now examine the result in more detail, focussing first on specifics questions concerning and EOIS and later on the procedure's strengths and weaknesses. We restrict these additional experiments to Chess, since it allows a greater amount of control (where appropriate, we have verified similar patterns on Job-Shop).

Recall versus precision

The results in Table 1 only tabulates recall numbers, which shows EOIS can identify more true positives than other methods. But how many other solutions does it generate, besides the true positives? (That is, what about its precision?) Table 2(a) tabulates the change in precision of EOIS (only shown for Chess: the behaviour for JobShop is similar), which shows precision improving with each iteration. This is consistent with the observation earlier that the numbers of negative examples increase with each iteration. The tabulations also suggest a relative gain in precision of about 3–9 over random sampling. This may not seem much, but can result in an order of magnitude less testing than random sampling, to obtain the same number of near-optimal instances. Precision can be improved, as usual, at the expense of recall. Table 2(b) shows this, achieved by generating fewer instances (100, as opposed to 1000) on each iteration.

Table 1 Estimates of number of near-optimal instances generated after each iteration of evolutionary optimisation

Model	Near-optimal instances		
	$k = 1$	$k = 2$	$k = 3$
(a) Chess			
SRS	1.0 (1.0)	2.0 (1.4)	3.0 (1.7)
	(1000)	(2000)	(3000)
EOIS	9.6 (7.6)	24.0 (1.6)	27.0 (0.0)
	(1000)	(2000)	(3000)
EODN	4.4 (1.2)	11.3 (2.3)	13.2 (1.7)
(DBN)	(1000)	(2000)	(3000)
EODN	5.1 (1.3)	12.4 (1.9)	14.8 (1.2)
(GAN)	(1000)	(2000)	(3000)
(b) Job-Shop			
SRS	3.0 (1.7)	6.0 (2.4)	9.0 (3.0)
	(1000)	(2000)	(3000)
EOIS	5.3 (2.9)	21.8 (9.5)	54.9 (0.3)
	(1000)	(2000)	(3000)
EODN	8.2 (0.6)	22.4 (2.3)	29.5 (2.9)
(DBN)	(1000)	(2000)	(3000)
EODN	15.8 (1.5)	27.2 (3.2)	39.1 (2.7)
(GAN)	(1000)	(2000)	(3000)

There are 27 near-optimal instnces overall in Chess and 304 in Job-Shop. Sampling variation in EOIS arises due to probabilistic selection of clauses when generating a model, and the entry is the mean value over 10 repetitions (the adjacent entry is the standard deviation). The number of samples generatd used is on the subsequent row. For SRS, the entries are simply obtained using binomial estimates, using the number of training instances used by EOIS. For the EODS procedures, the stochasticity inherent in the GANs and the DBNs result in sampling variations

Table 2 Estimated precision and recall of models (shown for Chess only)

Model	Precision and recall		
	$k = 1$	$k = 2$	$k = 3$
(a)			
SRS	0.001 0.037	0.001 0.074	0.001 0.111
EOIS	0.010 0.355	0.010 0.888	0.009 1.000
EOIS	Precision and recall		
sample	$k = 1$	$k = 2$	$k = 3$
(b)			
100	0.015 0.089	0.017 0.181	0.043 0.629
1000	0.005 0.355	0.008 0.888	0.009 1.000

Precision and recall are defined in the usual manner (i.e. $TP/(TP + FP)$; and $TP/(TP + FN)$ respectively, where TP, FP, FN denote numbers of true positives, false positives and false negatives). Here (a) refers to the models obtained by sampling $n = 1000$ instances per iteration (these are the models for the results are in Table 1. In (b) we show one way of improving precision at the cost of recall (by lowering the number of samples per iteration)

Table 3 Sensitivity to thresholds

EOIS	Threshold sequence		
	(8, 4, 0)	(8, 6, 0)	(12, 4, 0)
(a)			
Prec.	0.009	0.009	0.009
Recall	1.000	1.000	1.000

EOIS	Threshold sequence		
	(8, 4, 0)	(12, 0)	(8, 0)
(b)			
Prec.	0.009	0.001	0.008
Recall	1.000	0.074	0.592

All results refer to final models for the Chess problem. (a) Effect of small changes from an existing threshold sequence (the sequence in the first column (8, 4, 0) is the one used in experiments so far); (b) Effect of changing the number of thresholds

Threshold selection

The evolutionary optimisation procedure uses pre-defined thresholds, and two questions that naturally arise are: (1) How sensitive are the results are to the choices used?;and (2) Can threshold-selection be automated? Table 3(a) suggests that a different choice of starting or intermediate threshold does not make a significant difference to the precision or recall of the final model. However, the tabulation in Table 3(b) shows that the number and actual thresholds chosen can make a difference.

The results in Table 3 suggest a search procedure somewhat akin to that used by decision trees for identifying thresholds for continuous-valued attributes. Given a set of possible thresholds, the procedure simply searches for the next best threshold to apply, until no futher improvement is possible. By "best threshold" we mean here simply the threshold that results in identifying the most near-optimal instances. Given an upper-bound θ^* on the acceptable cost of solutions and ordered values $\theta_1 > \theta_2 > \cdots > \theta_n \theta^*$, the procedure can be implemented using EOIS with thresholds obtained from a greedy search through subsets of $\{\theta_1, \theta_2, \ldots, \theta_n, \theta^*\}$. One form of this greedy search starts with the set $\{\theta^*\}$; examines subsets $\{\theta_1, \theta^*\}$,$\{\theta_2, \theta^*\}$, $\ldots \{\theta_n, \theta^*\}$; finds the best subset, say $\{\theta_k, \theta^*\}$ and then proceeds to examine subsets $\{\theta_{k+1}, \theta_k, \theta^*\}$, $\{\theta_{k+2}, \theta_k, \theta^*\}$, $\ldots \{\theta_n, \theta_k, \theta^*\}$ and so on. Each sequence examined will require a model-construction step, and a sample-generation step. While the greedy approach attempts to minimise the number of models constructed, the samples generated can be fixed to a small number, since for most of the search, we are only interested in ordering sequences and not their exact value.

Table 4 shows that the sequence identified by this greedy procedure, which performs as well as the manual sequence for the same sample size ($n = 100$), but employs fewer thresholds. There is an additional cost arising from the need to examine multiple threshold sequences.

Role of domain knowledge

The performance of the ILP-assisted models in the Job-Shop domain is not as good as on Chess: at the end of 3 iterations, recall is only about 0.2, compared to 1.0 iin Chess. The

Table 4 Trace of a greedy procedure for selecting a sequence of thresholds from the set {0, 2, 4, 6, 8, 10, 12, 14} for Chess

Iteration	Threshold sequence	Near optimals
0	(0)	0
1	(14, 0)	2
	(12, 0)	1
	(10, 0)	2
	(8, 0)	**17**
	(6, 0)	6
	(4, 0)	1
	(2, 0)	0
2	(8, 6, 0)	7
	(8, 4, 0)	17
	(8, 2, 0)	17

The procedure starts with the sequence (0), and each iteration attempts to determine the best threshold to add to an existing sequence. With each sequence considered, EOIS is called with a sample size of 100, and the best sequence (with maximum near-optimals) is chosen. The procedure halts if none of the sequence in an iteration improve on the results from the previous iteration. The row marked in bold is the sequence returned

Table 5 Performance of the EOIS procedure with domain knowledge of low relevance to near-optimal solutions

Background	Near-optimal instances		
B	$k = 1$	$k = 2$	$k = 3$
SRS	3.0 (1.7)	6.0 (2.4)	6.1 (2.4)
B_{low}	5.3 (2.4)	5.0 (0.0)	5.0 (0.0)
	(1000)	(2000)	(2075)
B_{high}	9.6 (7.6)	24.0 (1.6)	27.0 (0.0)
	(1000)	(2000)	(3000)

The results are for Chess, with B_{low} denoting background predicates that simply define the geometry of the board, using the predicates $less\_than$ and $adjacent$. These predicates form the domain knowledge for most ILP applications to the problem of detecting illegal positions in the KRK endgame. B_{high} denotes background used previously, with high relevance to low depths-of-win. The numbers in parentheses are estimated standard deviations as before

natural question that arises is: Why is this so? We conjecture that this is a consequence of the domain knowledge. Recall that we had characterised the Job-Shop problem previously as being *knowledge-deficient*. That is, the domain knowledge used is not especially relevant to low-cost solutions. In the knowledge encoded for Chess, in contrast, some of the concepts refer specifically to “cornering” the Black King, with a view of ending the game as soon as possible. We would expect these predicates to be especially useful for positions at depths-of-win near 0. Evidence of the unreliable performance of the EOIS procedure in Chess can be obtained by removing such focused domain knowlege: see Table 5.

Table 6 Performance of deep networks with ILP-constructed rules as inputs

Model	Near-optimal instances		
	$k = 1$	$k = 2$	$k = 3$
DBN	4.4 (1.2)	11.3 (2.3)	13.2 (1.7)
DBN+ILP	5.3 (0.8)	12.3 (1.3)	18.7 (2.2)
GAN	5.1 (1.3)	12.4 (1.9)	14.8 (1.2)
GAN+ILP	4.3 (1.0)	14.7 (2.4)	19.1 (1.7)

At each iteration, values of rules in the corresponding EOIS theory are provided as (high-level) Boolean features, augmenting the (low-level) representation used by the deep network

The results in Table 5 suggest a refinement to the use of EOIS. Using the term "knowledge-rich domains" to signify domains with knowledge that is relevant to low-cost solutions:

We expect the EOIS procedure to be effective only in knowledge-rich domains.

The performance of EOIS also gives us a partial answer to the practical question of what is to be done when the relevance of the domain knowledge is unclear. Our results here suggest that if EOIS performance is comparable to simple random sampling, then this points to the domain knowledge being of low relevance.

Deep network performance

Ostensibly, the deep network models on both Chess and Job-Shop are worse. However this comparison is not appropriate. Recall that the network models are constructed without the benefit of the domain knowledge available to EOIS. We have seen above that EOIS performance can degrade significantly if domain knowledge is deficient. In light of the B_{low} results in Table 5, the performance of EODNs are in fact much better than EOIS, when domain knowledge is deficient (the deep networks only have access to the position of the pieces, which is even less than B_{low}). Using the term "knowledge-deficient domains" to signify problems where the domain knowledge is deficient or irrelevant for low-cost solutions, the results here suggest a further caveat on the use of EOIS:

We expect the EOIS procedure to be less effective than deep models in knowledge-deficient domains.

Incorporating domain knowledge into deep networks remains a topic of great interest. One simple way that can be used here is simply to provide the rules constructed by EOIS as input features to a deep network. Although simple, the approach nevertheless improves the performance of deep networks, although they still ro not reach the levels of EOIS with B_{high} (see Table 6).

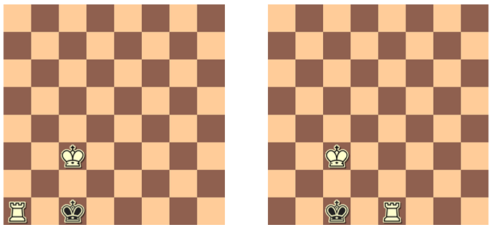
Comprehensibility

The logical representation used by EOIS allows us to inspect models constructed. This can be of some advantage for problems where it is important to understand how the near-optimal instances are generated. Figure 6 shows an example.

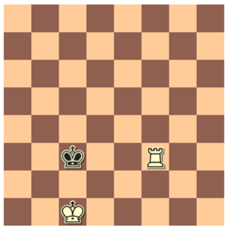
Finally, to correctly compare the value of model-assisted sampling against simple random sampling, we would need to first estimate the number of instances that need to be drawn

```
class(krk(WKF, WKR, WRF, WRR, BKF, BKR), good) :-
            canonical_rank_file(WKF,WKR,WRF,WRR,BKF,BKR),   % type enumerator
            file_distance(WKF, WRF, D1),  D1 >= 2,
            dist_to_closest_corner(WKF, WKR, D2), D2 =< 2,
            l_pattern(WKF, WKR, BKF, BKR, WRF, WRR).
```

(a)



(b)



(c)

Fig. 6 Understanding models. **a** A rule constructed for Depth = 0 (check-mate); **b** examples of good instances generated by the rule; and **c** a bad instance generated by the rule

randomly to obtain the same numbers of near-optimal instances as with model-assisted EDA. We then need to obtain two other costs: (a) the time to obtain values of the objective function for randomly-sampled instances; and (b) the time taken to obtain the corresponding values for the training data used to construct models plus the total time taken for model construction. For model-construction to be beneficial, clearly the time in (b) has to be less than (a). For the problems here, random sampling requires approximately 6 (Job-Shop) to 10 (Chess) times as many samples to obtain the same numbers of near-optimal instances as EOIS. The times for theory-construction are small enough to expect that (b) is less than (a) for these ratios.

What can be expected from EOIS on real datasets? Table 7 tabulates the performance on the two datasets described in Sect. 3.2. It is clear that EOIS does not appear to achieve anything useful on ADME, but why is this so? Based on the experimental results obtained with synthetic data, one reason could be that the domain knowledge provided for ADME is of little relevance to predicting very high solubility molecules such as the ones sought here (log$S$ values of ≥ 0). Further supportive evidence of this follows from the fact that the ILP engine was unable to find many patterns, even with low precision, for these solubility values. Unfortunately, we do not have the pharmocological expertise to provide additional domain knowledge to test this conjecture. For Malaria, it appears that the domain knowledge is relevant, but can possibly be enriched further.

4 Concluding remarks

This paper is concerned with model-driven generation of solutions for optimisation problems. Our interest is especially on hard problems for which there are no easy solutions, but there is significant, relevant human expertise. But how is such knowledge to be incorporated in the search for solutions? If the usual route of conversion into linear constraints is inappropriate, then it may be difficult to use numerical optimisation techniques efficiently. Dominance rules can help, of course, but these may not be easy to come by, especially for complex real problems. Equally difficult is to translate rules-of-thumb and problem requirements into a prior distribution function suitable for use by probabilistic modelling methods, So,

Table 7 Performance of EOIS on real data

Model	Near-optimal instances		
	$k = 1$	$k = 2$	$k = 3$
(a) ADME			
SRS	16.2 (3.9)	37.2 (5.9)	37.9 (5.9)
EOIS	17.2 (4.2)	35.0 (0.0)	35.0 (0.0)
	(250)	(565)	(575)
(b) Malaria			
SRS	12.7 (3.5)	17.1 (4.1)	21.5 (4.6)
EOIS	115.0 (0.0)	128.0 (4.6)	131.0 (0.0)
	(1000)	(1349)	(1693)

The results are the number of near-optimal instances after each iteration of evolutionary optimisation. As before, EOIS estimates are from 10 repetitions, and SRS entries are binomial estimates using the number of training instances used by EOIS (these are in parentheses in the row below the mean estimates for EOIS)

what options remain? One possibility is to use deep networks that attempt to re-construct the domain knowledge needed—in the form of intermediate features—from low-level data. At the other end of the spectrum is ILP, which provides one of the most flexible ways of incorporating domain knowledge in model construction. Our results suggest that if domain knowledge relevant to low-cost solutions is available, then ILP can make effective use of such knowledge.

ILP models to date have largely been discriminatory in nature. Generating answers is not new for logic programming: it has long been understood that a logic program can be used to compute more than "yes" or "no" answers. It is surprising therefore that logic programs constructed by ILP systems have rarely been used in a manner to enumerate instances. Instead, ILP models have mainly been used to decide if one or more instances are logically entailed, given some domain knowledge and the model constructed. This has not always been the case: early ILP systems like MARVIN (Sammut 1981), for example, did use theories constructed in a generative manner. Sampling from logic programs is also central to the modern strand of research in ILP on probabilistic Inductive Logic Programming (PILP: see De Raedt et al. 2008), and the use of ILP in this paper is akin to this form of generative modelling. Specifically, we see our use as rewriting the model constructed by a standard ILP engine as a stochastic logic program (SLP: Muggleton 1996) over which we allow backtrackable sampling in the manner described in Cussens (2000). This also points to the problem of generating near-optimal solutions as a useful application areafor methods combining probability and ILP (Riguzzi and Zese 2017; De Raedt et al. 2008). These methods are a result of research into probabilistic extensions to logic-programming (see: Riguzzi 2018; De Raedt and Kimmig 2015; Muggleton 1996; Sato and Kameya 1997), and on the interface of these methods to ILP (the PITA system, for example Riguzzi and Swift 2011). Here, domain knowledge and examples remain statements in definite-clause logic as required by standard ILP. But we know beforehand that instances can have labels attached to them, based on their (true) costs. Results in this paper suggest that relevance of domain knowledge can be important to constructing good generative models.

The performance of deep generative networks suggests that in knowledge-poor domains, they can still perform reasonably well. However, unless there is a large amount of data, we cannot expect deep networks to match ILP-based methods in knowledge-rich domains. Practitioners using deep networks have found interesting ways to overcome the small data problem, provided there is a related problem with very large amounts of data (a form of transfer-learning then becomes possible). But incorporating prior knowledge remains a challenge. The hybrid approach of using ILP to learn rules for low-cost solutions, and using these as Boolean features has shown promise here. This use of ILP for feature-construction has a long and productive history (Joshi et al. 2008; Saha et al. 2012; Srinivasan and King 1996; Ramakrishnan et al. 2007; Specia et al. 2009, 2006; Srinivasan et al. 2012), but their use in conjunction with deep networks has only just begun to be explored (Saikia et al. 2016; Dash et al. 2018), and may provide one way of drawing on the strengths of deep models and ILP for generating low-cost solutions for optimisation problems.

Acknowledgements AS is a Visiting Professorial Fellow, School of CSE, UNSW Sydney. AS is supported by the SERB Grant EMR/2016/002766. The authors would like to thank Sarmimala Sakia for her help with early versions of this work.

References

Angelopoulos, N., & Cussens, J. (2008). Bayesian learning of Bayesian networks with informative priors. *Annals of Mathematics and Artificial Intelligence*, *54*(1–3), 53. https://doi.org/10.1007/s10472-009-9133-x.

Bain, M. (1991). Experiments in non-monotonic learning. In *Proceedings of the eighth international workshop (ML91), Northwestern University, Evanston, Illinois, USA*, pp. 380–384.

Bain, M., & Muggleton, S. (1994). Learning optimal chess strategies. *Machine Intelligence*, *13*, 291–309.

Breda, G. (2006). *KRK chess endgame database. Knowledge extraction and compression* (T.U. Darmstadt, 2006). Diploma Thesis.

Cussens, J. (2000). Stochastic logic programs: Sampling, inference and applications. In *Proceedings of the sixteenth conference on uncertainty in artificial intelligence* (Morgan Kaufmann Publishers Inc., San Francisco, CA, USA), UAI'00, pp. 115–122. http://dl.acm.org/citation.cfm?id=2073946.2073961.

D'Ariano, A., Pacciarelli, D., & Pranzo, M. (2007). A branch and bound algorithm for scheduling trains in a railway network. *European Journal of Operational Research*, *183*, 643–657.

Dash, T., Srinivasan, A., Vig, L., Orhobor, O. I., & King, R. D. (2018). Large-scale assessment of deep relational machines. In *Proceedings of 28th international conference on inductive logic programming, ILP 2018*, Ferrara, Italy, September 2–4, 2018, pp. 22–37.

De Bonet, J. S., Isbell, C. L., & Viola, P. et al. (1997). MIMIC: Finding optima by estimating probability densities. In *Advances in neural information processing systems*, pp. 424–430.

De Raedt, L., Frasconi, P., Kersting, K., & Muggleton, S. (Eds.). (2008). *Probabilistic inductive logic programming—Theory and applications*, Lecture notes in computer science, Vol. 4911. Springer.

De Raedt, L., & Kimmig, A. (2015). Probabilistic (logic) programming concepts. *Machine Learning*, *100*(1), 5. https://doi.org/10.1007/s10994-015-5494-z.

Fischer, A., & Igel, C. (2012). An introduction to restricted Boltzmann machines. In *Iberoamerican congress on pattern recognition*, Springer, pp. 14–36.

Gamo, F. J., Sanz, L. M., Vidal, J., de Cozar, C., Alvarez, E., Lavandera, J. L., et al. (2010). Thousands of chemical starting points for antimalarial lead identification. *Nature*, *465*(7296), 305.

Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., & Bengio, Y. (2014). Generative adversarial nets. In *Advances in neural information processing systems*, pp. 2672–2680.

Harrod, S. (2010). A tutorial on fundamental model structures for railway timetable optimization. *Surveys in Operations Research and Management Science*, *17*, 85–96.

Hou, T., Xia, K., Zhang, W., & Xu, X. (2004). ADME evaluation in drug discovery. 4. Prediction of aqueous solubility based on atom contribution approach. *Journal of Chemical Information and Modeling*, *44*(1), 266. https://doi.org/10.1021/ci034184n.

Joshi, S., Ramakrishnan, G., & Srinivasan, A. (2008). Feature construction using theory-guided sampling and randomised search. In *ILP*, pp. 140–157.

Jouglet, A., & Carlier, J. (2011). Dominance rules in combinatorial optimization problems. *Operational Research*, *212*, 433.

King, R. D., Sternberg, M. J. E., & Srinivasan, A. (1995). Relating chemical activity to structure: An examination of ILP successes. *New Generation Computer*, *13*(3&4), 411. https://doi.org/10.1007/BF03037232.

Lloyd, J. W. (1987). *Foundations of logic programming* (2nd ed.). Berlin: Springer.

Muggleton, S. (1996). Stochastic logic programs. In *New generation computing*, Academic Press.

Muggleton, S., & Raedt, L. D. (1994). Inductive logic programming: Theory and methods. *Journal of Logic Programming*, *19,20*, 629.

Muggleton, S., Srinivasan, A., & Bain, M. (1992). Compression, significance, and accuracy. In *Proceedings of the ninth international workshop on machine learning (ML 1992)*, Aberdeen, Scotland, UK, July 1–3, 1992, pp. 338–347.

Pelikan, M., Goldberg, D. E., & Lobo, F. G. (2000). A survey of optimization by building and using probabilistic models. *Computational Optimization and Applications*, *21*, 5.

Ramakrishnan, G., Joshi, S., Balakrishnan, S., & Srinivasan, A. (2007). Using ILP to construct features for information extraction from semi-structured text. In *International conference on inductive logic programming*, Springer, pp. 211–224.

Riguzzi, F. (2018). *Foundations of probabilistic logic programming: Languages, semantics, inference and learning*. Valencia: River Publishers.

Riguzzi, F., & Swift, T. (2011). The PITA system: Tabling and answer subsumption for reasoning under uncertainty. *TPLP*, *11*(4–5), 433.

Riguzzi, F., & Zese, R. (2017). Probabilistic inductive logic programming on the web. In *Proceedings of the doctoral consortium, challenge, industry track, tutorials and posters @ RuleML+RR 2017 hosted by international joint conference on rules and reasoning 2017 (RuleML+RR 2017)*, London, UK, July 11–15, 2017.

Saha, A., Srinivasan, A., & Ramakrishnan, G. (2012). What kinds of relational features are useful for statistical learning? In F. Riguzzi & F. Zelezný (Eds.), *Inductive logic programming—22nd international conference, ILP 2012, Dubrovnik, Croatia, September 17–19, 2012, Revised selected papers*. Lecture notes in computer science (Vol. 7842, pp. 501–508). Springer

Saikia, S., Vig, L., Srinivasan, A., Shroff, G., Agarwal, P., & Rawat, R. (2016). Neuro-symbolic EDA-based optimization using ILP-enhanced DBNs. In *Proceedings of the workshop on cognitive computation: Integrating neural and symbolic approaches 2016 co-located with the 30th annual conference on neural information processing systems (NIPS 2016)*, Barcelona, Spain, December 9, 2016.

Sammut, C. (1981). Concept learning by experiment. In *Proceedings of the 7th international joint conference on artificial intelligence, IJCAI '81*, Vancouver, BC, Canada, August 24–28, 1981, pp. 104–105. http://ijcai.org/Proceedings/81-1/Papers/021.pdf.

Sato, T., & Kameya, Y. (1997). PRISM: A language for symbolic-statistical modeling. In *Proceedings of the fifteenth international joint conference on artificial intelligence, IJCAI 97*, Vol. 2, Nagoya, Japan, August 23–29, 1997, pp. 1330–1339.

Specia, L., Srinivasan, A., Joshi, S., Ramakrishnan, G., & Nunes, M. G. V. (2009). An investigation into feature construction to assist word sense disambiguation. *Machine Learning*, *76*(1), 109.

Specia, L., Srinivasan, A., Ramakrishnan, G., & Nunes, M. D. (2006). Word sense disambiguation using inductive logic programming. In *ILP*, pp. 409–423.

Srinivasan, A. (1999). *The Aleph manual*. http://www.comlab.ox.ac.uk/oucl/research/areas/machlearn/Aleph/.

Srinivasan, A. (2001). Extracting context-sensitive models in inductive logic programming. *Machine Learning*, *44*(3), 301.

Srinivasan, A., Faruquie, T., Bhattacharya, I., & King, R. (2012). Topic models with relational features for drug design. In *ILP*.

Srinivasan, A., & King, R. D. (1996). Feature Construction with inductive logic programming: A study of quantitative predictions of biological activity by structural attributes. In *6th international workshop on inductive logic programming, ILP-96*, Stockholm, Sweden, August 26–28, 1996, selected papers, pp. 89–104.

Srinivasan, A., King, R. D., Muggleton, S., & Sternberg, M. J. (1997). Carcinogenesis predictions using ILP. In *Proceedings of 7th international workshop inductive logic programming, ILP-97*, Prague, Czech Republic, September 17–20, 1997, pp. 273–287.

Affiliations

Ashwin Srinivasan[1] · **Lovekesh Vig**[2] · **Gautam Shroff**[2]

Lovekesh Vig
lovekesh.vig@tcs.com

Gautam Shroff
gautam.shroff@tcs.com

1 Department of Computer Science and Information Systems, BITS-Pilani, Goa Campus, Goa, India

2 TCS Research, New Delhi, India