AppliedMath

MDPI

Article

Solving Complex Optimisation Problems by Machine Learning †

Steven Prestwich

School of Computer Science and Information Technology, University College Cork, T12 XF62 Cork, Ireland; s.prestwich@cs.ucc.ie; Tel.: +353-36-21-420-5911

† This paper is an extended version of our paper published in the 9th International Conference on Machine Learning, Optimization, and Data Science, LOD 2023 (Grasmere, Lake District, UK, 22–26 September 2023).

Abstract: Most optimisation research focuses on relatively simple cases: one decision maker, one objective, and possibly a set of constraints. However, real-world optimisation problems often come with complications: they might be multi-objective, multi-agent, multi-stage or multi-level, and they might have uncertainty, partial knowledge or nonlinear objectives. Each has led to research areas with dedicated solution methods. However, when new hybrid problems are encountered, there is typically no solver available. We define a broad class of discrete optimisation problem called an influence program, and describe a lightweight algorithm based on multi-agent multi-objective reinforcement learning with sampling. We show that it can be used to solve problems from a wide range of literatures: constraint programming, Bayesian networks, stochastic programming, influence diagrams (standard, limited memory and multi-objective), and game theory (multi-level programming, Bayesian games and level-k reasoning). We expect it to be useful for the rapid prototyping of solution methods for new hybrid problems.

Keywords: multi-objective; multi-agent; reinforcement learning; optimisation

1. Introduction

Optimisation problems are ubiquitous in artificial intelligence, operations research, and a wide range of application areas. In their simplest form, they require us to make a set of decisions in order to optimise an objective function, possibly while satisfying some constraints. A decision is usually represented by the assignment of a domain value to a decision variable, where the domain might be a set of real numbers, integers, categories or data structures. Different domains, constraints and objectives are supported by different methods. Mixed integer linear programming (MILP) and more generally mathematical programming, constraint programming (CP), dynamic programming and metaheuristics (such as local search and genetic algorithms) can be applied to such problems.

However, many optimisation problems are more complex than this. In particular, they might involve uncertainty, which can be modelled by *chance variables* that are not controlled by a decision maker. Instead, they take values according to some probability distribution. Stochastic programming is an extension of MILP that models such problems. In multi-stage stochastic programs, decision and chance variables might be interleaved so that we must make decisions without knowing the values of some chance variables. Having made decisions, we can then observe some chance variables, but must then make next-stage decisions, and so on. A solution to a multi-stage problem is not a simple set of decisions, but a policy taking the form of a tree to allow reactions to random events. Probability distributions are usually assumed to be unaffected by decisions (*exogenous uncertainty*), but some applications have decision-dependent distributions (*endogenous uncertainty*). Stochastic dynamic programming is often used in the latter case.

Another possible complication is that a decision maker, or *agent*, might have multiple objectives: for example, to maximise profit while minimising environmental damage. Some form of compromise must be reached for such problems. Yet another complication is that a

check for updates

Citation: Prestwich, S. Solving Complex Optimisation Problems by Machine Learning. *AppliedMath* **2024**, *4*, 908–926. https://doi.org/10.3390/appliedmath4030049

Academic Editors: Libor Pekař and Jinyun Yuan

Received: 8 May 2024
Revised: 24 July 2024
Accepted: 29 July 2024
Published: 31 July 2024

problem might involve multiple agents. In multi-level programming, an agent must make a decision in the knowledge of how another agent will react, who might in turn have to take other agents into account. Knowledge might be partial: an agent might not know exactly how an adversary will act or has acted, and must make a decision based on assumptions.

Influence diagrams (IDs) are a particularly expressive formalism that can model multi-stage decision problems with endogenous uncertainty. Chance variables may be independent or related by conditional probability tables, forming a Bayesian network. *Observations* or the observed values of chance variables are also allowed, and Bayesian inference can be used to infer the distributions of unobserved chance variables based on these observations. Several exact and approximate methods are available for solving IDs.

A machine learning approach to complex optimisation problems is *reinforcement learning* (RL) [1] which can be used to solve multi-stage decision problems in stochastic environments. Multi-agent RL has famously been used to learn to play games such as Go and Chess to superhuman levels via self-play [2]. Multi-objective RL algorithms have also been devised, and these extensions have been combined in the field of multi-objective multi-agent RL. Similar developments have occurred in the evolutionary computation literature.

Thus, we have a wide range of available technologies, each able to solve certain types of problem efficiently. However, faced with a new optimisation problem with hybrid features, there might be no available solver. Some hybrids have been investigated, for example, multi-objective reinforcement learning with constraints [3], and bi-level multi-objective stochastic integer linear programming [4]. IDs have been extended to handle nonlinear utilities [5], multiple agents [6–10], multiple objectives [11,12], hard constraints [13], and partially observed variables [14]—though not all in one system. But no method exists for complex hybrids in general. To take a hypothetical example, there is no obvious way of modelling and solving a discrete bi-level optimisation problem whose leader is a non-linear chance-constrained stochastic program, and whose followers are a bi-objective constraint satisfaction problem and a weighted Max-SAT problem. A researcher faced with such a problem must invent a new approach or make simplifying assumptions to fit it into an existing framework. A general-purpose method able to handle all these complications would be very useful, at least for a preliminary investigation, though a more efficient specialised method might eventually be needed.

Another drawback of the fragmentation of technologies is that it is difficult to compare different approaches. Suppose we are faced with a new application with two decision makers but we are unsure how best to model it. Should we treat it as a bi-level program (see Section 3.8), a Bayesian game (see Section 3.9), using level-k thinking (see Section 3.10), model one decision maker as a random variable and implement a stochastic program (see Section 3.6) or take an expected value of one agent and reduce it to a constraint program (see Section 3.1)? Some simplifications might make little difference to the objective while others might make feasibility impossible. Each approach requires a different technology that takes considerable time to master, and possibly financial cost. A method that is general enough to solve all these models would be of great benefit for rapid prototyping, to explore the consequences of model design choices.

In this paper, we describe a very general class of discrete optimisation problem we call an *Influence Program* (IP) that generalises a wide range of problems from operations research, artificial intelligence and game theory (the name is inspired by the flexibility of IDs). We also present a simple solver for IPs based on a combination of multi-agent multi-objective reinforcement learning and Markov Chain Monte Carlo sampling. RL has been proposed as a unifying approach to sequential decision-making under uncertainty [15] and has solved complex adversarial games, making it a natural candidate for a general-purpose solver.

This paper extends our previous work. An early version using logic programming was described in [16], and a more recent C-based version appears in [17] where our problem class was called a "mixed influence diagram". This paper extends [17] by formalising the problem, providing more related work, replacing simple rejection sampling by Gibbs sampling, and solving a wider range of problems. Section 2 formalises the problem

and describes the algorithm. Section 3 applies it to a variety of problems from different optimisation literatures to demonstrate its flexibility and ease of use. Finally, Section 4 summarises the results and discusses future work.

2. The Influence Programming Framework

We now introduce our IP framework and an algorithm, after discussing related work.

2.1. Related Work

Considerable work has been performed on multi-objective RL (MORL). The MORL literature is too large to survey here, but a recent survey on MORL algorithms with a discussion on these issues is provided in [18]. Some methods use scalarisation to convert MORL into RL, making it a single-policy method like several others. Scalarisation has been criticised because it leads to only a single policy, and some MORL algorithms approximate Pareto fronts and learn policies for a range of weightings. Scalarisation has also been criticised on the grounds that it can be hard to choose appropriate weights, and that the choice puts decision power in the hands of engineers running the algorithm. Most such methods use linear scalarisation. Despite the shortcomings of scalarisation, we shall show that it can give good results on a range of problems. It is also easy to implement and has little runtime overhead, which is important for our lightweight approach. Results using weighted metrics are also less sensitive to the choice of weights [19].

Much work has also been performed on multi-agent RL (MARL), and a recent MARL survey can be found in [20]. Less work has been performed on the intersection of MARL and MORL: multi-objective multi-agent RL (MOMARL). A survey on MOMARL and related algorithms is provided in [21]. Because of the complexity of MOMARL problems, there is not even a single agreed definition of what constitutes a solution. MO-MIX [22] is a MOMARL algorithm using linear scalarisation, and deep RL in which an artificial neural network is used for state aggregation.

2.2. The Problem Class

We define a *(discrete) influence program* (IP) as a tuple $\langle \mathcal{V}, \mathcal{D}, \mathcal{A}, \mathcal{U}, \mathcal{L}, \mathcal{O}, \mathcal{P} \rangle$ where:

- $\mathcal{V} = \langle v_1, \ldots, v_n \rangle$ is an ordered list of variables v_i;
- $\mathcal{D} = \langle D_1, \ldots, D_n \rangle$ is a list of their corresponding finite domains D_i of possible values, typically ranges of integers or sets of symbolic names;
- $\mathcal{A} = \langle A_1, \ldots, A_n \rangle$ is a list of their corresponding *agents* (decision makers) A_i, which have symbolic names (in **bold**);
- $\mathcal{U}$ is a function mapping a total variable assignment to a utility vector for each agent;
- $\mathcal{L}$ is a set of directed links $v_i \to v_j$ $(i < j)$ between variables;
- $\mathcal{O}$ is a set of *observations*, where an observation is an assignment $v_i = d_j$ of a value $d_j \in D_i$ to a chance variable x_i;
- $\mathcal{P}$ is a function assigning a probability to each chance variable v_j assignment given an assignment for each variable v_i such that $v_i \to v_j \in \mathcal{L}$: it is typically represented by a (conditional) probability table.

Each variable is associated with exactly one agent, and we allow the possibility of chance variables whose agent has the name **chance**. The links $\mathcal{L}$ define which variables are visible to later decisions and chance variable distributions. A utility is a function mapping a total variable assignment (one value per variable) to a real value: we allow utilities to be *programmable*, requiring the user to write a small function to compute utilities from a total variable assignment. Each agent has at least one utility, apart from **chance**. The aim of an IP is to find a policy for each agent that Pareto-optimises its utilities in the context of observations and inter-variable visibility.

2.3. An Algorithm

We now describe the INFPROG algorithm shown in Algorithm 1. It is a lightweight research prototype using known techniques, used only to demonstrate the flexibility of our approach, and many other combinations of methods are potentially possible.

INFPROG is based on a simple RL algorithm: infinite-step tabular SARSA with ϵ-greedy action selection and learning rate α [1]. *Infinite-step* indicates that the reward is backed up equally to all values in the episode, which is more robust than Q-learning in the presence of unobserved variables [1]. *Tabular* indicates that state–action pairs have values in a table. The discount factor γ is set to 1 as our RL problem is episodic. If an agent has multiple utilities, these are scalarised so that there is one utility per agent (see Section 2.6). The scalarised objectives are used as rewards, computed at the end of each episode when all variables have been assigned updated values, and backed up to earlier states for those agents ($\mathbf{a}[v]$ denotes the agent $\mathbf{a}$ corresponding to variable v): the user must provide code for this step. However, the user is probably not interested in the value of the scalarised objective, so smoothed versions of the original objectives are printed out: these will be the values we report.

Algorithm 1 The INFPROG algorithm for solving IPs

Require: integers H, E and a utility hyperparameter Y
  initialise $\epsilon = 1$, $\alpha = 1$, $q_{H,v,i} = 0$
  for episode $e = 1 \dots E$ **do**
    for $v = v_1 \dots v_n$ **do**
      if v is chance **then**
        sample x_v from its distribution
      else if $U[0,1] < \epsilon$ **then**
        randomly sample $x_v \in D_v$
      else
        $x_v = \text{argmin}_{i \in D_v}(q_{H,v,i})$
      end if
    end for
    for each agent $\mathbf{a}$ **do**
      compute scalarised objective $f_{\mathbf{a}}(\vec{v})$ using Y
    end for
    for $v = v_1 \dots v_n$ **do**
      $q_{H,v,x_v} = \alpha f_{\mathbf{a}[v]}(\vec{v}) + (1-\alpha) q_{H,v,x_v}$
    end for
    update ϵ and α
  end for

The utility is computed at the end of an episode: in RL terms, this is a *sparse reward* which can make learning harder. In future work, we could allow for multiple value nodes as in IDs, or compensate for sparse rewards by adding RL techniques such as *hindsight experience replay* [23]. That technique was designed for off-policy RL algorithms but SARSA is on-policy, so some algorithm modification would be needed: for example, replacing SARSA by Q-learning or a more recent deep RL algorithm. There are many possibilities and this area is ripe for exploration.

The ϵ and α parameters start at 1 and decay to 0. Many decay schemes have been proposed in RL, and we arbitrarily choose $\alpha = \epsilon = ((E-e)/E)^3$. In our use of RL, a *state* is an assignment to variables $v_1 \dots v_i$ for some $i = 1 \dots n$, an *action* is the assignment of a value to a decision variable v_{i+1}, an *episode* assigns all the variables, and the *reward* is the utility (objective function value) computed at the end of an episode. The q_{H,v,x_v} are state–action values used to define the policy, which should optimise the expected reward.

Note that it is likely that for some problems it would be better to use SARSA with an eligibility trace and $0 < \lambda < 1$ (λ is a hyperparameter used with eligibility traces [1]) and we shall investigate this in future work. However, for our research, prototype we

effectively set $\lambda = 1$ by choosing infinite-step SARSA, thus simplifying the algorithm by removing a hyperparameter that requires tuning.

2.4. State Aggregation

Each decision might depend on all previous decisions and random events (but see Section 3.5), so a policy might involve a huge number of distinct states. To combat this problem, RL algorithms group together states via *state aggregation* methods. We choose a simple form called *random tile coding* [1], specifically *Zobrist hashing* [24] with H hash table entries for some large integer H. This works as follows. To each (decision or chance) variable–value pair $\langle v, x \rangle$, we assign a random integer r_{vx} which remains fixed. At any point during an episode, we have some set S of assignments $\langle v, x \rangle$, and we take the exclusive -or of the r_{vx} values (that is, their bit patterns) associated with assignments $X_S = \bigoplus_{\langle v,x \rangle \in S} r_{vx}$. Finally, we use X_S to index an array V with H entries: the value of q_{H,v,x_v} is stored in $V[X_S \bmod H]$ (in all our experiments, we fix $H = 2 \times 10^8$.)

The INFPROG algorithm takes two numerical hyperparameters: an integer H used for state aggregation and an integer E which is the number of iterations used by the solver. If H is sufficiently large then hash collisions are unlikely, and we will have a unique array element for each state encountered. It might be expected that Zobrist hashing will perform poorly when the number of states approaches or exceeds the size of the hash table, because hash collisions will confuse the values of different state–action pairs. Surprisingly, it can perform well even when hash collisions are frequent, and has been used in chess programming [25].

2.5. Sampling

INFPROG samples chance variables using a Markov chain Monte Carlo algorithm (MCMC) as in probabilistic programming. Thus, each SARSA episode is also a sweep through the chance variables. Our earlier work [16,17] used rejection sampling: at the end of a SARSA episode, if the chance variables did not match the observations, then the episode was rejected, in the sense that rewards were not backed up to earlier states. This worked on some problems but is impractical when probabilities are very low, and the use of an MCMC algorithm was proposed as future work. INFPROG uses Gibbs sampling and can handle such cases.

In the experiments, we found that some conditional probability table values that were set to 0 or 1 required adjustment: this is a known failure mode of Gibbs sampling. In fact, for optimisation problems without observations (which is true of most problems), we replace Gibbs with simple inverse transform sampling.

2.6. Multiple Objectives

Many real-world applications have multiple objectives and we must find a trade-off. Objectives can be combined in more than one way in RL, and a simple and popular approach is *linear scalarisation*: take a linear combination of the objectives, giving a single objective that can be used as an RL reward. This has the drawback that it cannot generate any solutions in a non-convex region of the Pareto front. It can also be hard to choose appropriate weights, especially when the objectives use different units.

Another approach is to rank the objectives by importance, then search for the lexicographically best result. This has the same problem as linear scalarisation (though recent work addresses this [26]) which can be fixed by thresholding all but the least important objective, a method called *thresholded lexicographic ordering* [27]. However, it has the drawback of using a specific RL algorithm. Moreover, in some of our applications, the most important objective is to minimise constraint violations, and this should not be thresholded.

The method we choose is to reduce the multiple objectives to a single objective via *weighted metric scalarisation* [19] in which the distance is minimised between the vector of values f_o for objectives o and a *utopian point* u^* in the multi-objective space:

$$\min_x \left(\sum_o w_o |f_o(x) - u_o^*|^p \right)^{1/p}$$

for some $p \geq 1$ and weights w_o chosen by the user. The need to choose weights is a disadvantage in terms of user-friendliness, but an advantage is that we can tune them to find different points on the Pareto front. The special case of *Chebyshev scalarisation* ($p = \infty$) is theoretically guaranteed to make the entire Pareto front reachable, but the L_p-norms ($p = 1, 2$) may perform better in practice [28]. In experiments, we found better results with L_1 than with L_2 or L_∞ despite Chebyshev's theoretical advantages, so we shall use L_1. However, in future work, it might be better to use Chebyshev and find ways of improving its results.

The utopian point is often adjusted during search to be just beyond the best point found so far, but INFPROG uses a fixed u^* provided by the user. Depending on whether each objective is to be maximised or minimised, we choose a value that is optimal or high/low enough to be unattainable.

Thus, to apply INFPROG to an optimisation problem, we must also provide a hyperparameter Y to guide scalarisation: a list of sublists of pairs. Each pair is a utopian value expressing a desired utility plus a weight, and each sublist corresponds to an agent and contains a pair for each of the agent's utilities. We shall always choose weights that sum to 1 for each agent, though this is not strictly necessary. The use of Y will be illustrated below via examples.

2.7. An Intuitive Explanation

To aid the reader, we now provide an intuitive explanation of how the algorithm works. The outer loop of the algorithm performs E *episodes* (a term taken from RL) where E is an integer hyperparameter chosen by the user. Each episode sweeps through the variables in the order specified by the list $\langle v_1, \ldots, v_n \rangle$, assigning each a value.

A decision variable is assigned a value as in the SARSA algorithm: during early episodes, assignment is largely random, but in later episodes, values are assigned more greedily in order to maximise the estimated expected *reward* (another RL term, corresponding to *utility* in influence diagrams and to *objective function value* in general optimisation). The degree of greediness is controlled by the ϵ parameter, which decays from 1 (completely random) in the first episode to 0 (completely greedy) in the last episode. In RL algorithms this strategy theoretically leads to an optimal policy.

A chance variable is assigned a value via sampling, thus an episode interleaves SARSA with sampling. In RL terms, the chance variable assignment is simply part of the environment in which the agent operates. In the special case where all variables are chance variables, the IP is a Bayesian network and an episode reduces to an MCMC sweep through the variables.

At the end of an episode, a reward is computed for each agent, and backed up to all earlier states as in SARSA, with the slight complication that in our multi-agent version each reward must be backed up to the appropriate agent.

The choice of infinite-step tabular SARSA was largely arbitrary, and has both advantages and disadvantages. An on-policy algorithm like SARSA is considered to be more consistent and stable than an off-policy algorithm like Q-learning, but also slower and less efficient. The choice of a tabular algorithm was made for its simplicity of implementation. The infinite-step choice corresponds to setting $\lambda = 1$ in an RL eligibility trace, making it a Monte Carlo method rather than a temporal difference method. The former are considered more robust in the presence of unobserved variables, as they are not based on the Markov property.

The restriction to rewards that occur only at the end of an episode was designed to simplify the implementation and user interface. In some cases, it would be preferable to allow intermediate rewards, if these occur naturally in the problem. However, this would require the user to write several pieces of code and to specify when each is to be executed. For our research prototype we took the simpler path.

Regarding the necessity for the user to provide programmable utility functions: although this might seem less user-friendly than some form of specification language, it is extremely flexible. We note that a similar approach is taken in the field of probabilistic programming [29,30].

3. Applications

In this section, we take small problems from a range of studies in the literature, model them as IPs, and solve them using InfProg. We shall not compare our method with others in terms of efficiency, as this is not the goal of the paper. For the same reason, we report few runtimes, though they are quite short—typically a few seconds and at most a few minutes. In fact, we do not expect it to be competitive on any particular class of problem, though as an RL-based method, it should perform reasonably well on applications from the RL literature. Our aim here is only to demonstrate that a single solver can solve a wide variety of optimisation problems, thus filling a gap in optimisation technology. We intend to implement faster, more scalable, and more user-friendly IP solvers in future work, and we hope that other researchers will also find better algorithms.

3.1. Constraint Programming

First we consider a "simple" optimisation problem: one agent and one objective. We take a well-known *constraint satisfaction problem* (CSP) known as eight queens: place eight chess queens on a standard 8×8 chessboard in such a way that no two queens attack each other (by being on the same row, column, or diagonal). This is a popular problem in constraint programming [31], and the smaller four queens problem is illustrated in Figure 1. The first example is a solution because no two queens are in the same row, column, or diagonal. The second example is a non-solution because it violates three constraints: the queens in the first two rows are in the same column, those in the last two rows lie on the same diagonal, and those in the first and third rows lie on another diagonal.

We treat eight queens as a *Max-CSP* problem in which the objective is to maximise the number of satisfied constraints (in this case, minimising the number of constraint violations to 0). We shall consider two possible IP models.

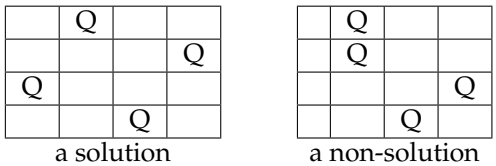


Figure 1. The 4-queens problem.

3.1.1. A Sparse Model

The problem can be modelled as follows: $\mathcal{V} = \langle v_1, \ldots, v_8 \rangle$, each $D_i = \{1, \ldots, 8\}$, $\mathcal{A} = \langle \mathbf{a}, \ldots, \mathbf{a} \rangle$ (all variables belong to the same agent **a**), $\mathcal{U} = \langle\langle viol \rangle\rangle$ where *viol* counts constraint violations:

$$\sum_{i=2}^{8} \sum_{j=1}^{i-1} [\![v_i = v_j \vee |i-j| = |v_i - v_j|]\!]$$

$[\![\cdot]\!]$ is the *Iverson bracket* that takes the value 1 if its argument is true and 0 if it is false. $\mathcal{P}$ is the empty function (there are no probability tables because there are no chance variables) and $\mathcal{L} = \mathcal{O} = \emptyset$. The utility hyperparameter is $\Upsilon = \langle\langle\langle 0, 1 \rangle\rangle\rangle$: one agent with one utility whose utopian point is 0 and associated weight is 1.

Though we shall not report runtimes in general, it might be of interest to the reader to see one example. Our solver is implemented in C and executed on a Intel(R) Core(TM) i5-6500 CPU @ 3.20 GHz with 8 GB of system RAM (Intel, Santa Clara, CA, USA). With 10^7 episodes, the solver takes 10.3 s per run, and in 100 runs, it found a correct solution 70 times. The other 30 times, the solution contained one constraint violation, illustrating that INFPROG can become trapped in local optima. It is also quite successful using only 10^6 episodes, finding 61 correct solutions.

This result is not competitive with methods such as constraint programming, integer programming or local search, which can all solve the problem more quickly and more reliably. But, as stated earlier, our aim is flexibility across a wide range of problems, not efficiency on any particular problem class.

3.1.2. A Dense Model

We referred to the previous model as *sparse* because it had few links (0). As an alternative *dense* model, we can add links to the IP from each decision variable to each earlier one: $\mathcal{L} = \{v_i \rightarrow v_j | i < j\}$. These are not logically necessary but they provide more information to the decision variables. With the dense IP, we found 100% success on eight queens using only 10^5 episodes, and even with only 30,000 episodes, it achieved 69% success. This suggests that making decisions visible to all subsequent decisions will improve performance. However, adding these links creates a number of states that are exponential in the number of variables, causing many hash collisions in INFPROG, which in turn might degrade performance.

This phenomenon is well known in RL. As an example, consider the elevator dispatching problem in which we must control a set of elevators in order to minimise the expected waiting times. Even if we restrict the problem to discrete time, it has a vast number of states, because of the many possible combinations of elevator positions, the direction of movement, and which buttons have been pressed. In principle, it would be best to treat every state separately, and the superior performance of our dense model supports this view, but combinatorial explosion makes it impractical for large problems.

One way of avoiding the explosion is to treat each elevator independently in a *distributed RL* approach—see, for example [32]. There is a decision-making agent for each elevator and they do not communicate, but they have a common objective and they indirectly learn to cooperate. A recent survey of distributed RL with applications is given in [33]. Our sparse IP works in the same way, though we only used one agent: we could instead use a separate agent for each variable, but it makes no difference because they have the same objective.

Controlling the links between variables makes it easy for INFPROG to emulate RL methods that are fully-, partially-, and non-distributed. For subsequent problems, we shall use a mixture of sparse and dense models.

3.2. Bayesian Networks

From a problem with all decision variables, we move to a problem with all chance variables: Pearl's alarm example [34]. This is an example of *probabilistic inference*, as performed in the field of probabilistic programming: inferring a conditional probability from a probabilistic program.

In this small problem, a house has an alarm that can be set off by an earthquake or a burglary, each with a prior probability. We also have conditional probabilities for the alarm under different circumstances: when there is/is not a burglary and there is/is not an earthquake. Given that the alarm has been activated, what is the probability that a burglary has occurred? The answer affects our actions.

We can model this as an IP. The variables are $\mathcal{V} = \{burglary, earthquake, alarm\}$, the domains $\mathcal{D}$ are all $\{yes, no\}$, the agents $\mathcal{A}$ are all **chance**, the links are $\mathcal{L} = \{burglary \rightarrow alarm, earthquake \rightarrow alarm\}$ as in the Bayesian network, the probabilities $\mathcal{P}$ are:

$$\begin{aligned} \Pr[burglary = yes] &= 0.7 \\ \Pr[earthquake = yes] &= 0.2 \\ \Pr[alarm = yes | burglary = yes, earthquake = yes] &= 0.9 \\ \Pr[alarm = yes | burglary = yes, earthquake = no] &= 0.8 \\ \Pr[alarm = yes | burglary = no, earthquake = yes] &= 0.1 \\ \Pr[alarm = yes | burglary = no, earthquake = no] &= 0.2 \end{aligned}$$

there is one observation $\mathcal{O} = \{alarm = yes\}$ and the utility vector is

$$\mathcal{U} = \langle\langle [\![earthquake = yes]\!] \rangle\rangle$$

Hence, the task is to compute the conditional probability $\Pr[earthquake = yes | alarm = yes]$. We can treat this as a utility, though the only agent is **chance** so we are not trying to optimise it by taking decisions. There are no decision-making agents, so the utility hyperparameter is $\mathrm{Y} = \langle\langle\langle\rangle\rangle\rangle$. InfProg applies Gibbs sampling and finds an expected "utility" of approximately 0.23, which is the correct probability.

We shall not use observations in subsequent examples as they are not a feature of most optimisation problems. However, they can be used to compute the *value of information* to determine which variables are most likely to reduce the uncertainty in a variable of interest.

3.3. Influence Diagrams

We now move to problems containing both decision and chance variables. IDs [35] are popular graphical models in decision analysis, and they can model important relationships between uncertainties, decisions, and values. They were initially conceived as tools for formulating problems but they have also emerged as efficient computational tools. They have many applications including medical diagnosis [36], cybersecurity [37], and risk management [38].

An ID is a directed acyclic graph with three types of node: *decision nodes* correspond to decision variables and are drawn as rectangles; *chance* (or *uncertain*) *nodes* correspond to chance variables and are drawn as ovals; *value nodes* correspond to preferences or objectives and are drawn as rounded rectangles, or polygons such as diamonds.

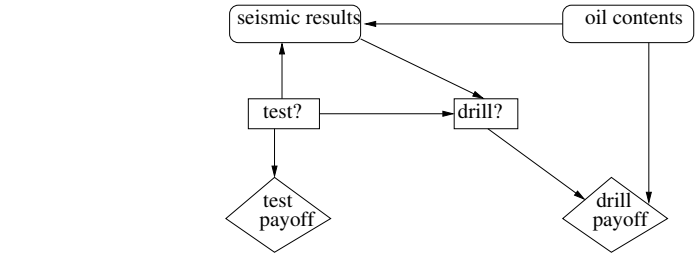
Each chance variable is associated with a conditional probability table that specifies its distribution for every combination of values for its parent variables in the graph. Each decision variable also has a set of parent variables in the graph, and its value depends only on their values. The decision variables are usually considered to be temporally ordered, and chance variables are observed at different points in the ordering. A standard ID assumption is *non-forgetting*, which means that the parents of any decision variable are all its ancestors in the graph: thus, a decision may depend on everything that has occurred before. All variables are discrete.

Each value node is associated with a table showing the *utility* (a real number) of each combination of parent variable values. A *decision policy* is a rule for each decision variable indicating how to choose its value from those of its parents. Any policy has a total *expected utility* (it is an expectation because of the chance variables) and *solving* an ID means computing its optimal policy, which has maximum expected utility.

Several methods exist for solving IDs. Some are exact and based on variable elimination [39–43] while others are approximate [44–48]. However, apart from our work ([16,17] and this paper), we know of no other applications of RL to solving IDs, which seems surprising as both IDs and RL can be used to model and solve sequential decision problems. The main connection usually made between the two is that IDs can model problems that can be tackled by RL, for example, causal IDs have been used to model artificial general intelligence safety frameworks which often use RL [49]. Our RL-based approach lies somewhere

between exact and approximate methods: given sufficient memory and training time, it has the potential to find an optimal solution but is not guaranteed to do so. We do not expect it to be as efficient as specialised algorithms, but it can tackle IDs with extensions such as limited memory (see Section 3.5), multiple agents, and multiple objectives. Moreover, the recent successes of deep RL make its application to large IDs a promising research direction. However, in this paper, we restrict ourselves to showing that RL can indeed solve IDs.

We use the Oil Wildcatter ID shown in Figure 2, a well-known problem published in [50]. An oil wildcatter must decide either to drill or not to drill for oil at a specific site. Before drilling, they may perform a seismic test that will help determine the geological structure of the site. The test result can be *closed* (indicating significant oil), *open* (indicating some oil), or *diffuse* (probably no oil). The special value *notest* means that test results are unavailable if the seismic test is not performed. The *test* decision does not depend on any other variable, but the *drill* decision depends on whether a test was made, and if so, on its result. The *oil* variable is unobservable so no decision depends on it (this is distinct from *forgetting* its value which is addressed in Section 3.5).



content	Pr[*O*]		
	dry	wet	soak
	0.5	0.3	0.2

test payoff	$U_1(T)$
test	
yes	−10
no	0

Seismic results		Pr[*S*|*O*, *T*]			
content	test	closed	open	diffuse	notest
dry	yes	0.1	0.3	0.6	0
dry	no	0	0	0	1
wet	yes	0.3	0.4	0.3	0
wet	no	0	0	0	1
soak	yes	0.5	0.4	0.1	0
soak	no	0	0	0	1

Drill payoff		$U_2(O, D)$
content	drill	
dry	yes	−70
dry	no	0
wet	yes	50
wet	no	0
soak	yes	200
soak	no	0

Figure 2. Oil wildcatter influence diagram.

An IP model is as follows. The variables are $\mathcal{V} = \langle oil, test, seismic, drill \rangle$, the domains are

$$\mathcal{D} = \langle \{dry, wet, soak\}, \{yes, no\}, \{closed, open, diffuse, notest\}, \{yes, no\} \rangle$$

the agents are $\mathcal{A} = \langle \textbf{chance}, \textbf{company}, \textbf{chance}, \textbf{company} \rangle$, the links are $\mathcal{L} = \{drill \rightarrow seismic, drill \rightarrow test\}$ (ID links to utilities are not part of an IP), there are no observations ($\mathcal{O} = \emptyset$), the utility vector is $\mathcal{U} = \langle \langle t + d \rangle \rangle$ where t is the test payoff and d is the drill payoff, and the probabilities are as shown in Figure 2. The utility hyperparameter is $\Upsilon = \langle \langle \langle 1000, 1 \rangle \rangle \rangle$: one agent with one objective to be maximised.

The known optimal policy given the above utilities and probabilities is as follows: apply the seismic test, and drill if the test result is open or closed. InfProg finds this solution and reports a close approximation to the correct expected utility: 22.5.

3.4. Multi-Objective Influence Diagrams

As a first example of multi-objective optimisation, we use a bi-objective oil wildcatter ID from [12]. In addition to maximising payoff, the aim is to minimise environmental damage. The IP is as in Section 3.3, except for additional payoffs: the utility of testing is now $(-10, 10)$ while the utility of drilling is $(-70, 18)$, $(50, 12)$, and $(200, 8)$ for dry, wet,

and soak, respectively: in each case, the first value is the original utility while the second value refers to the new utility. Instead of one optimal solution, there are four Pareto-optimal solutions, each with two utility values: (1) test, then drill if the result is closed or open, with utility $(22.5, 17.56)$; (2) do not test but drill, with the utility $(20, 14.2)$; (3) test, then drill if the result is closed, with utility $(11, 12.78)$; (4) do not test or drill, with utility $(0, 0)$.

The utilities are now $\mathcal{U} = \langle\langle\langle p, d\rangle\rangle\rangle$, where p is the payoff as before, and d is the environmental damage. With a utility hyperparameter $\mathrm{Y} = \langle\langle\langle 1000, 0.55\rangle, \langle -1000, 0.45\rangle\rangle\rangle$, in multiple runs, INFPROG found policies (1), (2), and (4), but not (3).

3.5. Limited Memory Influence Diagrams

Standard IDs are designed to handle situations involving a single, non-forgetful agent. Limited memory influence diagrams (LIMIDs) [14] are generalisations of IDs that allow decision making with limited information and simultaneous decisions, and can have much smaller policies. They relax the regularity (total variable ordering) and non-forgetting assumptions of IDs. LIMIDs are considered harder to solve optimally than IDs. We handle the limited memory feature in a simple way: during Zobrist hashing, the set S contains only assignments that are visible to the decision variable (as specified by the IP links $\mathcal{L}$).

For example, we use a pig breeding problem from [14]. A pig breeder grows pigs for four months and then sells them. During this period, the pig may or may not develop a disease. If it has the disease when it must be sold, then it must be sold for slaughter and its expected market price is 300. If it is disease-free, then its expected market price is 1000. Once a month, a veterinary surgeon tests the pig for the disease. If it is ill, then the test indicates this with a probability of 0.80, and if it is healthy, then the test indicates this with a probability of 0.90. At each monthly visit, the surgeon may or may not treat the pig, and the treatment costs 100. A pig has the disease in month 1 with a probability of 0.10. A healthy pig develops the disease in the next month with a probability of 0.20 without treatment and 0.10 with treatment. An unhealthy pig remains unhealthy in the next month with a probability of 0.90 without treatment, and 0.50 with treatment. The ID is shown in Figure 3.

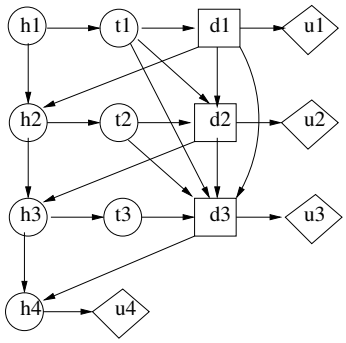


Figure 3. Pig breeding influence diagram.

An IP to model this problem is as follows. The variables are $\mathcal{V} = \langle h_i, t_i, d_i, \ldots, h_4\rangle$ $(i = 1, 2, 3)$, the domains $\mathcal{D}$ are $\{healthy, unhealthy\}$ for the h_i, $\{pos, neg\}$ for the t_i and $\{treat, leave\}$ for the d_i, there is one agent in $\mathcal{A}$ we call **breeder**, the links are $\mathcal{L} = \{h_i \to h_{i+1}, h_i \to t_j (i < j), t_i \to d_j (i \leq j), d_i \to h_{i+1}\}$, $\mathcal{P}$ contains the probabilities shown, $\mathcal{U} = \langle\langle u_1 + u_2 + u_3 + u_4\rangle\rangle$, and $\mathcal{O} = \emptyset$.

Using a utility hyperparameter $\mathrm{Y} = \langle\langle\langle 1000, 1\rangle\rangle\rangle$, INFPROG almost always finds the optimal policy with an expected utility of approximately 727: ignore the first test and do not treat, then follow the results of the other two tests (treat if positive). As noted in [16], a different policy is given in [14]: treat in month 3 if tests 1, and 2, or 3, are positive. We find that their policy has an expected utility of 725.884 while ours is optimal. They report the same expected utility as we do, so we believe this was simply a typographical error.

3.6. Multi-Stage Stochastic Programming

Stochastic programming [51] models and solves problems involving decision and chance variables, with known distributions for the latter. Problems may have one or more stages: in each stage, decisions are taken, then chance variables are observed. A solution is not a simple assignment of variables, but a policy that tells us how to make decisions given the assignments to variables from earlier stages. Stochastic programming dates back to the 1950s and is now a major area of research in mathematical programming and operations research.

We now show that discrete stochastic programs can be modelled and solved as IPs. We use a two-stage stochastic program from [52]:

$$\begin{array}{l}\max z = 1.5x_1 + 4x_2 + E[Q(x_1, x_2)] \\ \text{where } Q(x_1, x_2) \text{ is the value of} \\ \max 16y_1 + 19y_2 + 23y_3 + 28y_4 \text{ such that} \\ 2y_1 + 3y_2 + 4y_3 + 5y_4 \leq \xi_1 - x_1 \\ 6y_1 + y_2 + 3y_3 + 2y_4 \leq \xi_2 - x_2 \\ \text{where } 0 \leq x_1, x_2 \leq 5,\ y_i \in \{0, 1\}\ (i = 1, 2, 3, 4) \\ \xi_1, \xi_2 \text{ uniformly distributed on } \{5, 5.5, \ldots, 15\}\end{array}$$

A solution to this problem consists of fixed assignments to x_1, x_2, and assignments to the y_i that depend on the random ξ values. The optimum policy has an objective of 61.32.

We model the problem as an IP as follows. The variables are

$$\mathcal{V} = \langle x_1, x_2, \xi_1, \xi_2, y_1, y_2, y_3, y_4 \rangle$$

the agents are

$$\mathcal{A} = \langle \textbf{opt}, \textbf{opt}, \textbf{chance}, \textbf{chance}, \textbf{opt}, \textbf{opt}, \textbf{opt}, \textbf{opt} \rangle$$

(we choose the name **opt** for the non-random agent), the domains are

$$\mathcal{D} = \langle D_1, D_1, D_2, D_2, D_3, D_3, D_3, D_3 \rangle$$

where $D_1 = \{0, \ldots, 5\}$, $D_2 = \{5, 5.5, \ldots, 15\}$ and $D_3 = \{0, 1\}$, $\mathcal{P}$ assigns equal probabilities to the ξ_i domain values and $\mathcal{O} = \emptyset$.

As in Section 3.1, we model constraints by minimising their violations as an additional objective, choosing a large weight to prioritise feasibility. However, in this problem, we also have an objective so we shall use two utilities: the first to minimise the number of constraint violations to 0, and the second to maximise z. Hence, $\mathcal{U} = \langle \langle viol, z \rangle \rangle$. The utility hyperparameter is $\mathrm{Y} = \langle \langle \langle -10, 0.99 \rangle, \langle 100, 0.01 \rangle \rangle \rangle$.

3.6.1. A Sparse Model

In this IP, the links are $\mathcal{L} = \{\xi_i \rightarrow y_j | \forall i, j\}$. In multiple runs of up to 10^7 episodes, InfProg found policies with an expected utility of approximately 55. This is quite good but short of the known optimum of 61.32. Increasing to 10^9 episodes made no difference.

3.6.2. A Dense Model

In an attempt to improve the results, we added the following links to the sparse IP: $x_1 \rightarrow x_2$, $x_i \rightarrow y_j$ $(\forall i, j\}$ and $y_i \rightarrow y_j$ $(i < j)$. In multiple runs of 10^8 episodes, InfProg found policies with an expected reward approximately 60, which is close to the optimum.

3.7. Chance-Constrained Programming

A *chance* (or *probabilistic*) *constraint* is a constraint that should be satisfied with some probability threshold: $\Pr[C(x, r)] \geq \theta$ for a constraint C on decision variables x and chance variables r. Chance-constrained programming [53] is a method of optimising under uncertainty and has many applications, as it is a natural way of modelling uncertainty. Chance constraints are usually inequalities but we allow any form of constraint. Chance constraints

have also been used in the area of safe (or constrained) RL using various approaches [54,55], and our approach is related to that of [3]. They do not seem to have been added to IDs.

We model chance constraints by adding a new objective for each, with a reward of 1 for satisfaction, and 0 for violation, and setting the utopian value for that objective to the desired probability threshold. The weight attached to the objective should be high.

For example, we modify the stochastic program of Section 3.6 by attaching probabilities to the two hard constraints:

$$\Pr[2y_1 + 3y_2 + 4y_3 + 5y_4 \leq \xi_1 - x_1] = 0.7 \quad (a)$$
$$\Pr[6y_1 + y_2 + 3y_3 + 2y_4 \leq \xi_2 - x_2] = 0.9 \quad (b)$$

Notice that the chance constraints are of the form $\Pr[\cdot] = p$ instead of the usual $\Pr[\cdot] \geq p$. INFPROG is not guaranteed to satisfy the probability thresholds because of its multi-objective approach: instead of forcing the satisfaction of a chance constraint to exceed a threshold probability, it tries to match the threshold in a trade-off with other objectives. However, it can be used as an exploratory tool to find a policy with the desired characteristics, and the user can iteratively increase thresholds.

To model this problem, we modify the dense model from Section 3.6, adding a new objective for each chance constraint. The utility vector is now $\mathcal{U} = \langle\langle viol_a, viol_b, z\rangle\rangle$. The utility hyperparameter is

$$\mathrm{Y} = \langle\langle\langle 0.7, 0.496\rangle, \langle 0.9, 0.496\rangle, \langle 100, 0.008\rangle\rangle\rangle$$

To solve this tri-objective IP, we experimented with different weights and found a variety of policies, with different compromises between the chance constraints and original objective. Not all solutions were useful, but using weights $(0.496, 0.496, 0.008)$, we found a policy with objectives $(0.74, 0.95, 71)$ that satisfies the requirements. Relaxing the hard constraints to chance constraints allowed us to increase the objective z from 60 to 71.

3.8. Multi-Level Programming

Many problems in economics, diplomacy, war, politics, industry, gaming, and other areas involve multiple agents, which form part of each other's environment. Multi-agent RL can be applied to these problems, as can several forms of ID: bi-agent IDs [7], multi-agent IDs [8], game theory-based IDs [10], networks of IDs [6], and interactive dynamic IDs [9]. LIMIDs can model multiple agents, but only the cooperative case in which they all have the same objective.

The most common case is a *bi-level program* or *Stackelberg game*. These usually involve continuous variables and relatively little work has been performed on discrete bi-level programs [56], so new methods are of interest. However, as an even more challenging case, we use a discrete *tri-level program* studied in at least two publications [57,58] and shown in Figure 4.

$\max_{x_1} z_1 = (x_1 + x_2 + 2x_3 + 4)(-x_1 - x_2 + x_3 + 2x_4 + 1)$
where x_2 solves
$\max_{x_2} z_2 = 2x_2 + x_3 + 3x_4$
where $x_3, x_3, x_3, x_3, x_3, x_3$ solve, for given x_1, x_2
$\max_{x_3 \ldots x_8} z_3 = \frac{2x_1 + 3x_2 + 2x_3 - 3x_4}{5x_1 + 11x_2 + x_5 + 29}$
subject to
$-3x_1 + 7x_2 + x_3 + x_5 = 10 \quad 14x_1 + 4x_2 + x_6 = 6$
$x_1 + x_2 + x_3 - x_4 + x_7 = 5 \quad 2x_1 + x_2 + 2x_4 + x_8 = 8$
$0 \leq x_1 \leq 5 \quad 0 \leq x_2 \leq 2 \quad 5 \leq x_3 \leq 20 \quad 4 \leq x_4 \leq 30$
$1 \leq x_5 \leq 20 \quad 0 \leq x_6 \leq 30 \quad 0 \leq x_7 \leq 15 \quad 0 \leq x_8 \leq 20$

Figure 4. A tri-level non-linear program.

Tri-level programs have been used to model problems in supply chain management, network defence, planning, logistics, and economics. They involve three agents: the first is the *top-level leader*, the second is the *middle-level*) follower, and the third is the (*bottom-level*) follower. They are organised hierarchically: the leader makes decisions first, the

middle-level follower reacts, then the bottom-level follower reacts. A survey on multi-level programming is provided in [59], including a section on the tri-level case. Decentralised decision-making problems in a hierarchical management system often contain more than two levels. As a different application, we mention [60] which uses tri-level programming to model a defender–attacker–defender problem in defending an electrical power grid.

This problem is also of interest because its objectives are quadratic for the leader, linear for the middle-level follower, and fractional for the bottom-level follower. Non-linear objectives were added to IDs in [5] using non-linear optimal control approximations.

We model the problem using the following IP. The variables are $\mathcal{V} = \langle x_1, \ldots, x_8 \rangle$, and their domains are ranges of integer values $\mathcal{D} = \langle 0-5, 0-2, 5-20, 4-30, 1-20, 0-30, 0-15, 0-20 \rangle$. The agents are $\mathcal{A} = \langle \textbf{leader}, \textbf{middle}, \textbf{bottom}, \ldots, \textbf{bottom} \rangle$: v_1 is a top-level leader variable, v_2 is a middle-level leader variable, and v_3–v_8 are follower variables. The links are $\mathcal{L} = \{x_i \to x_j | i < j\}$ (a dense model in the sense of Section 3.1), the utilities are

$$\mathcal{U} = \langle \langle viols, z_1 \rangle, \langle viols, z_2 \rangle, \langle viols, z_3 \rangle \rangle$$

and $\mathcal{O} = \emptyset$. The first objective used by all agents is the number of constraint violations, which should be 0: the constraints ensure that any solution is *tri-level feasible* so all agents should be penalised for violations (in this problem, no agent is allowed to make a decision that leads to a constraint violation). The secondary objectives are z_1, z_2, z_3. Using a utility hyperparameter

$$\mathrm{Y} = \langle \langle \langle 0, 0.99999 \rangle, \langle 1000, 0.00001 \rangle \rangle, \langle \langle 0, 0.999 \rangle, \langle 100, 0.001 \rangle \rangle, \langle \langle 0, 0.99 \rangle, \langle 1, 0.01 \rangle \rangle \rangle$$

This is the most complex Y value we used, so we provide some explanation. The trilevel problem has been modelled using three agents, each with two objectives (one for the problem objectives and one to maximise constraint satisfaction). We use weighted metric scalarisation to reduce this to three agents with one objective each, so we must specify three utopian points (one per agent). Each utopian point has two ideal objective values and an associated weight for each value. So, for example, agent 1 has a utopian point $\langle 0, 0.99999 \rangle$ with associated weights $\langle 1000, 0.00001 \rangle$.

In repeated runs with 10^7 episodes, InfProg reliably finds the correct policy (0, 0, 9, 4, 1, 6, 0, 0) with objectives $((0, 396), (0, 21), (0, 0.2))$. As pointed out in [17], the first objective is given as 612 in [58], but it can be verified that the solution yields $z_1 = 396$. The suboptimal solutions that it sometimes finds can be recognised by their lower z_1 values.

3.9. Bayesian Games

The assumptions of classical game theory require complete information, which is often unrealistic. Bayesian games were a development that allowed incomplete (private or secret) information while avoiding infinite calculations [61]. An important feature is that a player may have a "type" that is not known by other players, representing their state of mind.

As a simple example, we take a well-known problem called the *Sheriff's dilemma*. There are two players: a sheriff and an armed suspect. The suspect has two possible types, a criminal or a civilian, and only he knows what type he is. The sheriff must know whether to shoot without knowing the suspect's type, while the suspect is allowed to use that information. Payoff matrices for the various possibilities are shown in Figure 5, where in each case, the first figure is a suspect payoff and the second a sheriff payoff.

We can model this as an IP as follows. The variables are $\mathcal{V} = \langle type, suspect, sheriff \rangle$ with domains

$$\mathcal{D} = \langle \{criminal, civilian\}, \{shoot, not\}, \{shoot, not\} \rangle$$

the agents are $\mathcal{A} = \langle \textbf{chance}, \textbf{suspect}, \textbf{sheriff} \rangle$. The links are $\mathcal{L} = \{type \to suspect\}$ allowing the suspect but not the sheriff to know the suspect's type. The type probabilities $\mathcal{P}$ are p for $\Pr[criminal]$ and $1 - p$ for $\Pr[civilian]$, the utilities are $\mathcal{U} = \langle \langle ususpect \rangle, \langle usheriff \rangle \rangle$, and $\mathcal{O} = \emptyset$.

Using a utility hyperparameter $\Upsilon = \langle\langle\langle 100, 1\rangle\rangle, \langle\langle 100, 1\rangle\rangle\rangle$ and 1000 episodes, INFPROG reliably finds the correct policy: the suspect shoots if and only if he is a criminal, and the sheriff shoots if and only if $p > \frac{1}{3}$. If p is close to $\frac{1}{3}$, then more episodes are needed to reduce sampling error.

type: criminal		sheriff action: shoot	sheriff action: not
suspect action	shoot	0,0	2,−2
	not	−2,−1	−1,1

type: civilian		sheriff action: shoot	sheriff action: not
suspect action	shoot	−3,−1	−1,−2
	not	−2,−1	0,0

Figure 5. Payoffs for the Sheriff's Dilemma.

3.10. Level-k Reasoning

Bayesian games have also been criticised for making unrealistic assumptions that do not always predict real-world behaviour. Another game theory approach to incomplete information is *level-k reasoning* [62,63], which assumes that players play strategically but with bounded rationality. It has recently been used in *adversarial risk analysis* for modelling problems in counter-terrorism [64] and other fields.

We consider a simple level-k problem: the *Keynesian beauty contest*. Contest participants are asked to choose a number that they hope will be as close as possible to some fraction p of the average of all participants' choices. Classical game theory predicts that all players will choose 0, which is the Nash equilibrium. The reasoning is that, if all players choose randomly, it is best to choose p times the mean: $50p$ or approximately 33 when $p = \frac{2}{3}$ (which is a popular value). But all players know this, so they should instead choose p times that value approximately 25. But all players know this, so the logical conclusion is that 0 is the only reasonable choice, yet in experiments, few people choose 0.

In level-k reasoning, we make assumptions about the other players. We might assume that all other players are level-0 thinkers who do not think strategically. They simply choose a number uniformly at random: $c_0 \in U[0, 100]$ (we shall restrict our numbers to be integers as our method is discrete). Under this assumption, we should use level-1 thinking and choose p times the expected level-0 choice: 33. If we instead assume that all other players are level-1, we should use level-2 thinking and choose 22. Similarly, if we assume that all others are level-2, we should use level-3 thinking and choose 15.

We can model and solve these problems as IPs, as we now show using the beauty contest. The variables represent the choices of the hypothetical level-k players (k $= 0, 1, 2$) and the real level-3 player: $\mathcal{V} = \langle c_0, c_1, c_2, c_3\rangle$. The domains in $\mathcal{D}$ are all $\{0, 1, 2, \ldots, 100\}$. The agents are $\mathcal{A} = \langle$**chance**, **P1**, **P2**, **P3**$\rangle$. The **chance** probabilities are all $\frac{1}{101}$. There are no links as all choices are made simultaneously, without knowing the other choices: $\mathcal{L} = \emptyset$. There are no observations so $\mathcal{O} = \emptyset$. The level-i player should guess as close as possible to p times the value they expect from the level-$(i-1)$ player, i.e., $|c_i - pc_{i-1}|$ should be minimised, so the utilities are $\langle\langle|c_1 - pc_0|\rangle, \langle|c_2 - pc_1|\rangle, \langle|c_3 - pc_2|\rangle\rangle$. Applying INFPROG with 10^6 episodes and a utility hyperparameter $\Upsilon = \langle\langle\langle 0, 1\rangle\rangle, \langle\langle 0, 1\rangle\rangle, \langle\langle 0, 1\rangle\rangle\rangle$ to minimise all utilities, we find c_1, taking values approximately 29–37, c_2 19–25 and c_3 13–15. Using more episodes reduces the variance in these numbers.

3.11. Practical Implications

We have demonstrated that a wide range of problems from diverse areas of literature can be modelled as IPs, and that these can be solved by the INFPROG solver. Of course, we do not claim that *any* problem from Operations Research, Game Theory and so on can be tackled by our approach, especially as it can currently handle only discrete variables. However, we believe that it has interesting practical implications.

Firstly, our approach is useful for rapid prototyping. Few researchers are masters of stochastic programming, dynamic programming, game theory, machine learning, and multiple other areas. When faced with a new complex optimisation problem, it is not always clear how to model it, and in practice, we often make simplifying assumptions based on our background. A stochastic programmer might simplify a multi-agent problem

by modelling an adversary using random variables, or pretend that decisions cannot affect chance variable distributions, or approximate a multi-stage problem by a two-stage one. A constraint programmer or integer programmer might approximate a chance variable by an expectation. A game theorist might assume a zero-sum game or complete knowledge, even when this is unrealistic. In general, it is hard to know whether our design choices are reasonable. We believe that a tool such as INFPROG is useful during the early stages of modelling, as it enables us to explore the consequences of various simplifications without the need to master multiple research areas: we can simply change the problem formulation and observe the result. Some simplifications might cause little change in solution quality, while others might greatly reduce quality or even make a problem infeasible. The eight-queens and stochastic program models can be seen as examples: in both cases, we implemented different models and compared the resulting objective values, leading to the conclusion that dense models give better results.

Secondly, we hope that our approach will be useful in its own right. It is based on multi-agent multi-objective RL, a very general paradigm with wide applications. For specific problems, we do not expect INFPROG to be competitive with specialised algorithms, but for problems that are inherently complex, it might be a useful tool, as there is little available software able to tackle problems involving multiple agents, multiple objectives, random events, and partial knowledge.

It should be noted that INFPROG is merely a research prototype and we certainly do not consider it to be a finished product. We hope that more competitive IP algorithms will emerge in the future, especially based on deep RL and implemented on highly parallel hardware. This should greatly enhance the scalability of the IP approach to complex optimisation problems, as RL has already shown its ability to tackle large multi-agent problems such as learning to play Go.

3.12. A Small Case Study

To illustrate our approach in more detail, we show the programmable utility for the Oil Wildcatter of Section 3.3 problem in Figure 6. The user must provide a C function called `utility` with two arguments: a one-dimensional array `v` of integer chance and decision variables and a two-dimensional array `u` of utilities. This function is called at the end of an episode, so we can assume that all variables in `v` have been assigned values. (The `*` and `**` are C notations indicating array dimensionality).

```
void utility(int* v, float** u)
{
   int oil=v[0], test=v[1], seismic=v[2], drill=v[3],
       no=0, yes=1, dry=0, wet=1, soak=2, company=1;
   float drill_payoff, test_payoff;

   if (test==yes) test_payoff= -10.0; else test_payoff=0.0;

   if (oil==dry  && drill==yes) drill_payoff= -70.0; else
   if (oil==wet  && drill==yes) drill_payoff=  50.0; else
   if (oil==soak && drill==yes) drill_payoff= 200.0; else
                                drill_payoff=   0.0;

   u[company][0]=drill_payoff+test_payoff;
}
```

Figure 6. Programmable utility for the oil wildcatter example.

For readability, we have created integer variables to name the IP variables (for example `oil=v[0]`), their possible values (for example `yes=0`) and the only agent (`company`—note that chance variables are assigned agent number 0). The total payoff is the sum of the drill and test payoffs, and this sum is assigned to the only utility `u[1][0]` in the problem. The second index is `0` because an agent's utilities are numbered from 0: for a bi-objective problem such as that in Section 3.4, we would also need to assign a value to `u[company][1]`.

The user must also provide a small text file describing the parameters of the problem (number of agents, variables, domain sizes, and so on). We do not show an example as INFPROG is currently a research prototype and lacks a user-friendly specification language.

4. Conclusions

This paper makes two contributions. Firstly, it defines a new class of optimisation problem called an influence program (IP) that is sufficiently general to encompass a wide variety of problems from operations research, artificial intelligence, game theory, and adversarial risk analysis. Secondly, it proposes an algorithm called INFPROG based on reinforcement learning plus sampling. INFPROG is a lightweight solver that does not need a graphics processing unit or other specialised hardware. We demonstrated the flexibility of IP using examples taken from diverse literatures: to the best of our knowledge, no other solver can tackle this range of optimisation problems. Our work also shows that many existing problems from other literatures can serve as benchmark problems for multi-objective and/or multi-agent reinforcement learning algorithms, of which there is a lack [18].

We expect IPs to be useful for hybrid optimisation problems that do not fit well into any particular class and have no available solvers. A fruitful source of applications might be safe reinforcement learning, in which constraints are used to ensure robust and safe policies. It might also be useful for the rapid prototyping of different models, to investigate how to model a new problem, to test the effects of hiding information, or making simplifying assumptions. Finally, it could be used as an educational tool for understanding the different models of optimisation problems.

In future work, we shall explore the application of our method to complex real-world problems. The current INFPROG implementation is a research prototype only, intended to demonstrate the potential of a unified approach to a wide range of complex optimisation problems. It can handle only discrete variables, uses a simple and quite weak form of state aggregation (random tile coding) and lacks a clear user-interface. We hope to design a more user-friendly version that also includes continuous variables, and to use more powerful state aggregation: the latter two improvements might be achieved by moving from SARSA to an actor–critic algorithm such as proximal policy optimisation [65].

Funding: This material is based upon works supported by the Science Foundation Ireland under Grant No. 12/RC/2289-P2 which is co-funded under the European Regional Development Fund. We would also like to acknowledge the support of the Science Foundation Ireland CONFIRM Centre for Smart Manufacturing, Research Code 16/RC/3918, and the EU H2020 ICT48 project TAILOR under contract #952215.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: No new data were created or analyzed in this study. Data sharing is not applicable to this article.

Conflicts of Interest: The funders had no role in the design of the study; in the collection, analyses, or interpretation of data; in the writing of the manuscript; or in the decision to publish the results.

References

1. Sutton, R.S.; Barto, A.G. *Reinforcement Learning: An Introduction*; MIT Press: Cambridge, MA, USA, 1998.
2. Silver, D.; Hubert, T.; Schrittwieser, J.; Antonoglou, I.; Lai, M.; Guez, A.; Lanctot, M.; Sifre, L.; Kumaran, D.; Graepel, T.; Lillicrap, T.; Simonyan, K.; Hassabis, D. A General Reinforcement Learning Algorithm That Masters Chess, Shogi, and Go Through Self-Play. *Science* **2018**, *362*, 1140–1144. [CrossRef] [PubMed]
3. Huang, S.H.; Abdolmaleki, A.; Vezzani, G.; Brakel, P.; Mankowitz, D.J.; Neunert, M.; Bohez, S.; Tassa, Y.; Heess, N.; Riedmiller, M.A.; Hadsell, R. A Constrained Multi-Objective Reinforcement Learning Framework. In Proceedings of the 5th Conference on Robot Learning, London, UK, 8–11 November 2021; pp. 883–893.
4. Elshafei, M.M.K.; El-Sherberry, M.S. Interactive Bi-level Multiobjective Stochastic Integer Linear Programming Problem. *Trends Appl. Sci. Res.* **2008**, *3*, 154–164.
5. Kratochvíl, V.; Vomlel, J. Influence Diagrams for Speed Profile Optimization. *Int. J. Approx. Reason.* **2017**, *88*, 567–586. [CrossRef]

6. Gal, Y.; Pfeffer, A. Networks of Influence Diagrams: A Formalism for Representing Agents' Beliefs and Decision-Making Processes. *J. Artif. Intell. Res.* **2008**, *33*, 109–147. [CrossRef]
7. González-Ortega, J.; Insua, D.R.; Cano, J. Adversarial Risk Analysis for Bi-agent Influence Diagrams: An Algorithmic Approach. *Eur. J. Oper. Res.* **2019**, *273*, 1085–1096. [CrossRef]
8. Koller, D.; Milch, B. Multi-Agent Influence Diagrams for Representing and Solving Games. *Games Econ. Behav.* **2001**, *45*, 181–221. [CrossRef]
9. Polich, K.; Gmytrasiewicz, G. Interactive Dynamic Influence Diagrams. In Proceedings of the 6th International Joint Conference on Autonomous Agents and Multiagent Systems, Communications in Computer and Information Science, Honolulu, HI, USA, 14–18 May 2007; Volume 288, pp. 623–630.
10. Zhou, L.H.; Kevin, L.; Liu, W.Y. Game theory-based Influence Diagrams. *Expert Syst.* **2013**, *30*, 341–351. [CrossRef]
11. Diehl, M.; Haimes, Y. Influence Diagrams With Multiple Objectives and Tradeoff Analysis. *IEEE Trans. Syst. Man Cybern. Part A* **2004**, *34*, 293–304. [CrossRef]
12. Marinescu, R.; Razak, A.; Wilson, N. Multi-objective Influence Diagrams. In Proceedings of the 28th Conference on Uncertainty in Artificial Intelligence, Catalina Island, CA, USA, 5–17 August 1 2012; pp. 574–583.
13. Jenzarli, A. Information/Relevance Influence Diagrams. In Proceedings of the 11th Conference on Uncertainty in Artificial Intelligence (UAI), Montreal, QC, Canada, 18–20 August 1995; pp. 329–337.
14. Lauritzen, S.L.; Nilsson, D. Representing and Solving Decision Problems With Limited Information. *Manag. Sci.* **2001**, *47*, 1238–1251. [CrossRef]
15. Powell, W.B. *Reinforcement Learning and Stochastic Optimization: A Unified Framework for Sequential Decisions*; Wiley: Hoboken, NJ, USA, 2022.
16. Prestwich, S.D.; Toffano, F.; Wilson, N. A Probabilistic Programming Language for Influence Diagrams. In Proceedings of the 11th International Conference on Scalable Uncertainty Management, Granada, Spain, 4–6 October 2017.
17. Prestwich, S.D. Solving Mixed Influence Diagrams by Reinforcement Learning. In Proceedings of the 9th International Conference on Machine Learning, Optimization, and Data Science, Grasmere, UK, 22–26 September 2023.
18. Hayes, C.F.; Rădulescu, R.; Bargiacchi, E.; Källström, J.; Macfarlane, M.; Reymond, M.; Verstraeten, T.; Zintgraf, L.M.; Dazeley, R.; Heintz, F.; et al. A Practical Guide To Multi-Objective Reinforcement Learning and Planning. *Auton. Agent. Multi-Agent Syst.* **2022**, *36*, 26. [CrossRef]
19. van Moffaert, K.; Drugan, M.M.; Nowé, A. Scalarized Multi-Objective Reinforcement Learning: Novel Design Techniques. In Proceedings of the IEEE Symposium on Adaptive Dynamic Programming and Reinforcement Learning, Singapore, 16–19 April 2013; pp. 191–199.
20. Zhang, K.; Yang, Z.; Başar, T. Multi-Agent Reinforcement Learning: A Selective Overview of Theories and Algorithms. In *Handbook of Reinforcement Learning and Control*; Vamvoudakis, K.G., Wan, Y., Lewis, F.L., Cansever, D., Eds.; Studies in Systems, Decision and Control; Springer: Cham, Switzerland, 2021; Volume 325.
21. Rădulescu, R.; Mannion, P.; Roijers, D.M.; Nowé, A. Multi-Objective Multi-Agent Decision Making: A Utility-Based Analysis and Survey. *Auton. Agents Multi-Agent Syst.* **2020**, *34*, 10. [CrossRef]
22. Hu, T.; Luo, B.; Yang, C.; Huang, T. MO-MIX: Multi-Objective Multi-Agent Cooperative Decision-Making with Deep Reinforcement Learning. *IEEE Trans. Pattern Anal. Mach. Intell.* **2023**, *45*, 12098–12112. [CrossRef] [PubMed]
23. Andrychowicz, M.; Wolski, F.; Ray, A.; Schneider, J.; Fong, R.; Welinder, P.; McGrew, B.; Tobin, J.; Abbeel, P.; Zaremba, W. Hindsight Experience Replay. In Proceedings of the 31st International Conference on Neural Information Processing Systems, Long Beach, CA, USA, 4–9 December 2017; pp. 5055–5065.
24. Zobrist, A.L. A New Hashing Method with Application for Game Playing. *ICGA J.* **1990**, *13*, 69–73.
25. Hyatt, R.M.; Cozzie, A. The Effect of Hash Signature Collisions in a Chess Program. *ICGA J.* **2005**, *28*, 131–139. [CrossRef]
26. Skalse, J.; Hammond, L.; Griffin, C.; Abate, A. Lexicographic Multi-Objective Reinforcement Learning. In Proceedings of the 31st International Joint Conference on Artificial Intelligence, Vienna, Austria, 23–29 July 2022; pp. 3430–3436.
27. Gábor, Z.; Kalmár, Z.; Szepesvári, C. Multi-Criteria Reinforcement Learning. In Proceedings of the 15th International Conference on Machine Learning, Madison, WA, USA, 24–27 July 1998; pp. 197–205.
28. Giagkiozis, I.; Fleming, P.J. Methods for Multi-Objective Optimization: An Analysis. *Inf. Sci.* **2015**, *293*, 338–350. [CrossRef]
29. Gordon, A.D.; Henzinger, T.A.; Nori, A.V.; Rajamani, S.K. Probabilistic Programming. In Proceedings of the International Conference on Software Engineering, Hyderabad, India, 31 May–7 June 2014.
30. Pfeffer, A. *Practical Probabilistic Programming*; Manning Publications: Shelter Island, NY, USA, 2016.
31. Rossi, F.; van Beek, P.; Walsh, T. (Eds.) *Handbook of Constraint Programming*; Elsevier: Amsterdam, The Netherlands, 2006.
32. Crites, R.H.; Barto, A.G. Elevator Group Control Using Multiple Reinforcement Learning Agents. *Mach. Learn.* **1998**, *33*, 235–262. [CrossRef]
33. Useng, M.; Avdulrahman, S. A Survey on Distributed Reinforcement Learning. *Mesopotamian J. Big Data* **2022**, *2022*, 44–50. [CrossRef]
34. Pearl, J. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*; Series in Representation and Reasoning; Morgan Kaufman Publishers: Burlington, MA, USA, 1988.
35. Howard, R.A.; Matheson, J.E. Influence Diagrams. In *Readings in Decision Analysis*; Strategic Decisions Group: Menlo Park, CA, USA, 1981; Chapter 38, pp. 763–771.

36. Nease, R.F., Jr.; Owens, D.K. Use of Influence Diagrams to Structure Medical Decisions. *Med. Decis. Mak.* **1997**, *17*, 263–275. [CrossRef] [PubMed]
37. Chockalingam, S.; Maathuis, C. Influence Diagrams in Cyber Security: Conceptualization and Potential Applications. In Proceedings of the 22nd European Conference on Cyber Warfare and Security, Piraeus, Greece, 22–23 June 2023; Volume 22.
38. Carriger, J.F.; Newman, M.C. Influence Diagrams as Decision-Making Tools for Pesticide Risk Management. *Integr. Environ. Assess. Manag.* **2011**, *8*, 339–350. [CrossRef]
39. Dechter, R. A New Perspective on Algorithms for Optimizing Policies Under Uncertainty. In *Artificial Intelligence Planning Systems*; Association for the Advancement of Artificial Intelligence: Washington, DC, USA, 2000; pp. 72–81.
40. Jensen, F.; Jensen, V.; Dittmer, S. From Influence Diagrams to Junction Trees. In *Uncertainty in Artificial Intelligence*; Morgan Kaufmann: Burlington, MA, USA, 1994; pp. 367–373.
41. Shachter, R.D. Evaluating Influence Diagrams. *Oper. Res.* **1986**, *34*, 871–882. [CrossRef]
42. Shenoy, P. Valuation-Based Systems for Bayesian Decision Analysis. *Oper. Res.* **1992**, *40*, 463–484. [CrossRef]
43. Tatman, J.A.; Shachter, R.D. Dynamic Programming and Influence Diagrams. *IEEE Trans. Syst. Man Cybern.* **1990**, *20*, 365–379. [CrossRef]
44. Cano, A.; Gómez, M.; Moral, S. A Forward-Backward Monte Carlo Method for Solving Influence Diagrams. *Int. J. Approx. Reason.* **2006**, *42*, 119–135. [CrossRef]
45. Charnes, J.M.; Shenoy, P.P. Multistage Monte Carlo Method for Solving Influence Diagrams Using Local Computation. *Manag. Sci.* **2004**, *50*, 405–418. [CrossRef]
46. Marinescu, R.; Lee, J.; Dechter, R. A New Bounding Scheme for Influence Diagrams. In Proceedings of the 35th Conference on Artificial Intelligence, Vancouver, BC, Canada, 2–9 February 2021; Association for the Advancement of Artificial Intelligence: Washington, DC, USA, 2021; pp. 12158–12165.
47. Watthayu, W. Representing and Solving Influence Diagram in Multi-Criteria Decision Making: A Loopy Belief Propagation Method. In Proceedings of the International Symposium on Computer Science and its Applications, Hobart, TAS, Australia, 13–15 October 2008; pp. 118–125.
48. Yuan, C.; Wu, X. Solving Influence Diagrams Using Heuristic Search. In Proceedings of the International Symposium on Artificial Intelligence and Mathematics, Sanya, China, 23–24 October 2010.
49. Everitt, T.; Kumar, R.; Krakovna, V.; Legg, S. Modeling AGI Safety Frameworks with Causal Influence Diagrams. In Proceedings of the Workshop on Artificial Intelligence Safety, CEUR Workshop, Honolulu, HI, USA, 27 January 2019; Volume 2419.
50. Raiffa, H. *Decision Analysis*; Addison-Wesley: Reading, MA, USA, 1968.
51. Birge, J.R.; Louveaux, F.V. *Introduction to Stochastic Programming*; Springer: New York, NY, USA, 2011.
52. Ahmed, S.; Tawarmalani, M.; Sahinidis, N.V. A Finite Branch-and-Bound Algorithm for Two-Stage Stochastic Integer Programs. *Math. Program.* **2004**, *100*, 355–377. [CrossRef]
53. Charnes, A.; Cooper, W.W. Chance-Constrained Programming. *Manag. Sci.* **1959**, *6*, 73–79. [CrossRef]
54. García, J.; Fernández, F. A Comprehensive Survey on Safe Reinforcement Learning. *J. Mach. Learn. Res.* **2015**, *16*, 1437–1480.
55. Gu, S.; Yang, L.; Du, Y.; Chen, G.; Walter; Wang, J.; Yang, Y.; Knoll, A.C. A Review of Safe Reinforcement Learning: Methods, Theory and Applications. *arXiv* **2022**, arXiv:2205.10330.
56. Kovacs, A.; Kis, T. Constraint Programming Approach to a Bilevel Scheduling Problem. *Constraints* **2011**, *16*, 317–340. [CrossRef]
57. Arora, R.; Arora, S.R. An Algorithm for Non-Linear Multi-Level Integer Programming Problems. *Int. J. Comput. Sci. Math.* **2010**, *3*, 211–225. [CrossRef]
58. Mishra, S.; Verma, A.B. A Non-Differential Approach for Solving Tri-Level Programming Problems. *Am. Int. J. Res. Sci. Technol. Math.* **2015**.
59. Lu, J.; Han, J.; Hu, Y.; Zhang, G. Multilevel Decision-Making: A Survey. *Inf. Sci.* **2016**, *346–347*, 463–487. [CrossRef]
60. Alguacil, N.; Delgadillo, A.; Arroyo, J.M. A Trilevel Programming Approach for Electric Grid Defense Planning. *Comput. Oper. Res.* **2014**, *41*, 282–290. [CrossRef]
61. Harsanyi, J.C. Games with Incomplete Information Played by "Bayesian" Players, I–III: Part I. The Basic Model. *Manag. Sci.* **2004**, *50*, 1804–1817. [CrossRef]
62. Nagel, R. Unraveling in Guessing Games: An Experimental Study. *Am. Econ. Rev.* **1995**, *85*, 1313–1326.
63. Stahl, D.O.; Wilson, P.W. On Players' Models of Other Players: Theory and Experimental Evidence. *Games Econ. Behav.* **1995**, *10*, 218–254. [CrossRef]
64. Rothschild, C.; Albert, L.; Guikema, S. Adversarial Risk Analysis with Incomplete Information: A Level-k Approach. *Risk Anal.* **2011**, *32*, 1219–1231. [CrossRef]
65. Schulman, J.; Wolski, F.; Dhariwal, P.; Radford, A.; Klimov, O. Proximal Policy Optimization Algorithms. *arXiv* **2017**, arXiv:1707.06347.