Solving Mixed Influence Diagrams by Reinforcement Learning

S. D. Prestwich

School of Computer Science and Information Technology, University College Cork, Ireland
Email: s.prestwich@cs.ucc.ie

Abstract. While efficient optimisation methods exist for problems with special properties (linear, continuous, differentiable, unconstrained), real-world problems often involve inconvenient complications (constrained, discrete, multi-stage, multi-level, multi-objective). Each of these complications has spawned research areas in Artificial Intelligence and Operations Research, but few methods are available for hybrid problems. We describe a reinforcement learning-based solver for a broad class of discrete problems that we call Mixed Influence Diagrams, which may have multiple stages, multiple agents, multiple non-linear objectives, correlated chance variables, exogenous and endogenous uncertainty, constraints (hard, soft and chance) and partially observed variables. We apply the solver to problems taken from stochastic programming, chance-constrained programming, limited-memory influence diagrams, multi-level and multi-objective optimisation. We expect the approach to be useful on new hybrid problems for which no specialised solution methods exist.

1 Introduction

Optimisation problems are ubiquitous in artificial intelligence, operations research and a wide range of application areas. They occur in many forms, each with a range of specialised modelling languages and solution methods: mixed integer linear programming (MILP), constraint programming (CP), stochastic programming, chance-constrained programming, dynamic programming, bilevel programming and others. Some hybrids have been investigated — for example multi-objective reinforcement learning with constraints [19] and bilevel multi-objective stochastic integer linear programming [9] — and CP and MILP have been extended in various directions: stochastic optimisation, robust optimisation, multiple objectives and multiple decision makers. However, no method exists for complex hybrids in general. For example there is no obvious way of solving a bi-level optimisation problem whose leader is a non-linear stochastic program with chance constraints, 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.

Influence diagrams (IDs) [18] are a particularly expressive formalism that can model multi-stage decision problems, in which an agent makes decisions by assigning values to decision variables, observes the (possibly random) results, then makes further decisions, and so on depending on the number of stages. Chance variables model uncertainties and are not controlled by a decision-maker. They may be independent or related by conditional probability tables as in Bayesian networks (which are a special case of IDs without decisions or utilities). Decision variables may appear in these tables and thus affect chance variable distributions, a feature sometimes called *endogenous uncertainty* as opposed to the more common *exogenous uncertainty* where distributions are independent of decisions. Conversely, decisions may depend on chance variable assignments.

IDs have been extended in several ways, and in this paper we combine many of them into a problem class we call Mixed Influence Diagrams (MIXIDs) which allow:

- *Non-linear utilities.* In fact we allow programmable utilities.
- *Multiple agents*. Each decision variable may belong to any one of a set of agents (decision makers) each with its own utility. Agents may be independent, adversarial or cooperative.
- *Multiple objectives.* Each agent may have more than one utility.
- *Constraints.* Many optimisation problems are constrained, and we allow hard, soft and chance constraints.
- *Partially observed variables.* We allow chance and decision variables to be observable or unobservable by subsequent decision variables.

Each of these extensions have been added to IDs in earlier work, at least to some extent (see below), but to the best of our knowledge they have not been combined into a single framework. The contribution of this paper is to show that many problems can be modelled as MIXIDs and solved using a reinforcement learning-based method called MIXID1. In Section 2 we describe MIXID1 and provide references to previous ID extensions. In Section 3 we apply it to a variety of problems from different optimisation literatures to demostrate its flexibility and ease of use. In Section 4 we summarise the results and discuss future work.

2 Modelling and solving MIXIDs

We now introduce the MIXID class of problems and the MIXID1 solver. First we introduce standard IDs.

2.1 Influence diagrams

IDs 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 for the evaluation stage.

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. An example of an ID is shown in Section 3.1.

2.2 The basic algorithm

We now describe a reinforcement learning (RL) approach to solving standard IDs, similar to that described in [32]. Suppose we have a single agent with one objective f to be minimised: a standard ID. Each variable v_i $(i = 1, \ldots, n)$ is either a decision variable or a chance variable. The variables are strictly ordered $v_1 < \ldots < v_n$ according to their temporal ordering. Denote the current assignment of v by x_v. Each variable v_i has a domain size s_v indicating that it takes values from its domain $D_v = \{0, 1, \ldots, s_v - 1\}$.

```
initialise ϵ = 1, α = 1 and q_{v,i} = 0
for episode e = 1 ... E
    update ϵ and α
    for v = v_1 ... v_n
        if v is chance
            sample x_v from its distribution
        else with probability ϵ
            randomly sample x_v ∈ D_v
        else
            x_v = argmin_{i∈D_v}(q_{v,i})
    compute objective f(v⃗)
    for v = v_1 ... v_n
        q_{v,x_v} = αf + (1 − α)q_{v,x_v}
```

Fig. 1. The MIXID1 algorithm for standard IDs

The basic algorithm is shown in Figure 1. It is based on a simple RL algorithm: infinite-step tabular SARSA with ϵ-soft action selection and learning rate α [36]. The RL reward is the objective of the optimisation problem (the ID utility) which is computed at the end of each episode when all variables have been assigned updated values. The user must provide code for this step. Note that in our IDs there is only one value node, which includes all those values that would usually be computed earlier. This simplifies the implementation without loss of generality. *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 [36]. *Tabular* indicates that state-action pairs have values in a table. The discount factor γ is set to 1 as the RL problem is episodic.

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 decision variable v_{i+1}, an *episode* assigns all the variables, and the *reward* is the objective value computed at the end of an episode. The q_{v,x_v} are state-action values used to define the policy, which should optimise the expected reward.

Each decision depends on all previous decisions and random events (but see Section 2.3) so a policy might involve a huge number of distinct states. To combat this problem, RL algorithms group together states via *state aggregation* methods. For our prototype method we choose a simple form called *random tile coding* [36], specifically *Zobrist hashing* [41] 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 it, 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_{v,x_v} is stored in $V[X_S \bmod H]$. 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 [20].

2.3 Partially observed variables

Standard IDs are designed to handle situations involving a single, non-forgetful agent. Limited memory influence diagrams (LIMIDs) [25] are generalizations 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 those assignments that are visible to the decision variable (its parent variables, indicated by a link in the ID graph).

2.4 Multiple objectives

Many real-world applications have multiple objectives and we must find a trade-off. Multi-objective (or multi-criteria) IDs were described in [8, 28]. 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 a RL reward. This has the drawback that it can not 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 [35]) which can be fixed by thresholding all but the least important objective, a method called *thresholded lexicographic ordering* [11]. 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* [30] 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 [14] and we found best results with L_1.

The utopian point is often adjusted during search to be just beyond the best point found so far, but MIXID1 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.

2.5 Hard and soft constraints

Many optimisation problems have constraints that must be met by any solution. Constraints are not modelled in standard IDs but hard constraints were supported in information/relevance IDs [21]. We model soft constraints by minimising their violations as an additional objective. To model hard constraints we use the popular technique of choosing a large weight to prioritise minimising violations to zero.

2.6 Chance constraints

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 [5] 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) Reinforcement Learning using various approaches [13, 17], and our approach is related to that of [19]. Chance constraints do not seem to have been added to IDs in previous work.

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.

2.7 Multiple agents and non-linear objectives

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 [16], multi-agent IDs [23], game theory-based IDs [40], networks of IDs [12] and interactive dynamic IDs [15]. LIMIDs can model multiple agents, but only the cooperative case in which they all have the same objective.

We extend MIXID1 to allow multiple agents numbered $1, 2, \ldots$ Each decision variable v belongs to an agent a_v, and we use 0 as a dummy agent number for chance variables (sometimes called a *chance agent*). Each agent $a \geq 1$ has its own objective f^a, each of which is computed at the end of an episode. The algorithm is modified so that each reward f^a is backed up only to values q_{v,x_v} such that $a_v = a$. This transforms MIXID1 into a multi-agent RL algorithm.

3 Experiments

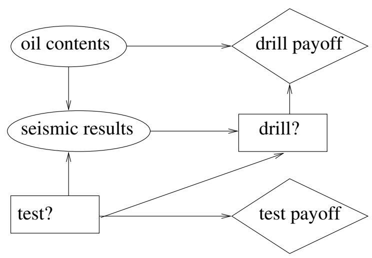
We now illustrate the flexibility of the method by applying it to a variety of problems taken from diverse literatures. We do not compare it with standard methods in terms of efficiency, as this is not the goal of the paper. For the same reason we do not report runtimes, though they are quite short: typically a few seconds and at most a few minutes. In fact we do not expect MIXID1 to be competitive on any particular class of problem, though as a RL method it should perform well on some applications. 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. In future work we will explore more efficient RL methods, especially using deep neural networks for state aggregation.

3.1 A standard ID

As a first example we use the well-known Oil Wildcatter ID shown in Figure 2. 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 done. There are two decision variables Test (T) and Drill (D), and two chance variables Seismic (S) and Oil (O).

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 2.3). The payoff tables model costs and benefits as utilities: the test payoff is negative (a cost) if the test is performed, otherwise zero; the drill payoff may be positive if oil is found, negative if not, and zero if no drilling occurs. In our implementation the two tables are computed after all variables are assigned values.

We use probabilities and utilities as given in [28]. The known optimal policy for this problem is: apply the seismic test, and drill if the test result is open or closed. MIXID1 finds this solution which has expected utility 22.5.



oil contents	$P(O)$		
	dry	wet	soak
	0.5	0.3	0.2

test payoff	$U_1(T)$
Test?	
yes	-10
no	0

Seismic results		$P(S \mid O, T)$			
Oil cnt.	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		$U_2(O, D)$
Oil cnt.	Drill?	
dry	yes	-70
dry	no	0
wet	yes	50
wet	no	0
soak	yes	200
soak	no	0

Fig. 2. The oil wildcatter ID

3.2 Partially observed variables

As an example we use a pig breeding problem from [25]. A pig breeder grows pigs for four months 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 test the pig for the disease. If it is ill then the test indicates this with probability 0.80, and if it is healthy then the test indicates this with probability 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 probability 0.10. A healthy pig develops the disease in the next month with probability 0.20 without treatment and 0.10 with treatment. An unhealthy pig remains unhealthy in the next month with probability 0.90 without treatment, and 0.50 with treatment.

In the LIMID shown in Figure 3 the h are random variables denoting healthy or unhealthy, the t are random variables denoting positive or negative test results, the d are decision variables denoting treat or leave, the u are utilities, and the numbers 1–4 denote month. The pig breeder does not keep detailed records, which are forgotten, and bases treatment decisions only on the previous test result for each pig. MIXID1 almost always finds the optimal policy with 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).

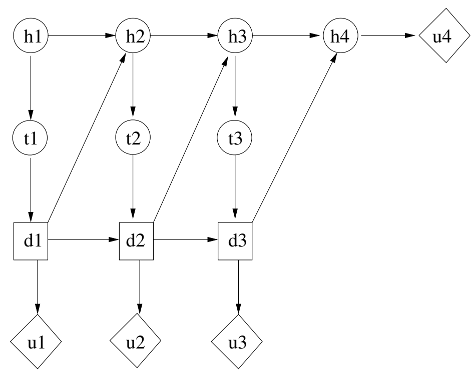


Fig. 3. The pig breeding LIMID.

3.3 Multiple objectives

We use the bi-objective oil wildcatter ID in [28]. In addition to maximising payoff the aim is to minimise environmental damage. 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 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 $(0, 0)$. In multiple runs MIXID1 found policies (1), (2) and (4) using a utopian point (1000,-1000) and weights (0.55,0.45), though it did not find policy (3).

3.4 Hard and soft constraints

We take an example from stochastic programming [3] which 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 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 use a 2-stage stochastic program from [1]:

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

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 objective 61.32 with $(x_1, x_2) = (0, 4)$.

We model the problem as a MIXID with two objectives: the first to minimise the number of constraint violations to 0, and the second to maximise z. Decision variables can observe all ancestor variables. We use a utopian point $(-10, 100)$ and weights $(0.99, 0.01)$: note that $0.99 \gg 0.01$, chosen because the constraints are hard. In multiple runs MIXID1 found policies with (x_1, x_2) equal to $(0, 4)$ or $(0, 3)$ and expected reward approximately 60.

3.5 Chance constraints

We modify the stochastic program of Section 3.4 by attaching probabilities to the two hard constraints:

$$\begin{array}{l}\Pr[2y_1 + 3y_2 + 4y_3 + 5y_4 \leq \xi_1 - x_1] = 0.7 \\ \Pr[6y_1 + y_2 + 3y_3 + 2y_4 \leq \xi_2 - x_2] = 0.9\end{array}$$

Notice that the chance constraints are of the form $\Pr[\cdot] = p$ instead of the usual $\Pr[\cdot] \geq p$. MIXID1 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.

As described in Section 2.5 these chance constraints are implemented by adding a new objective for each, but this time with utopian point $(0.7, 0.9, 100)$. To solve this tri-objective MIXID 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 z from 60 to 71.

3.6 Multiple agents and non-linear objectives

We use a *tri-level program* studied in at least two publications [2, 29] and shown in Figure 4. 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. Many algorithms have been published on tri-level (also bi- and multi-level) programs, but mostly on linear continuous problems and relatively little work has been done on discrete problems.

$\max_{x_1} z_1 = (x_1 + x_2 + 2x_3 + 4)(-x_1 - x_2 + x_3 + 2x_4 + 1)$
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
$\max_{x_3 \dots x_8} z_3 = \frac{2x_1+3x_2+2x_3-3x_4}{5x_1+11x_2+x_5+29}$
subject to

$$\begin{array}{llll}
-3x_1 + 7x_2 + x_3 + x_5 = 10 & & 14x_1 + 4x_2 + x_6 = 6 & \\
x_1 + x_2 + x_3 - x_4 + x_7 = 5 & & 2x_1 + x_2 + 2x_4 + x_8 = 8 & \\
0 \le x_1 \le 5 & 0 \le x_2 \le 2 & 5 \le x_3 \le 20 & 4 \le x_4 \le 30 \\
1 \le x_5 \le 20 & 0 \le x_6 \le 30 & 0 \le x_7 \le 15 & 0 \le x_8 \le 20
\end{array}$$

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

This problem is also of interest because its objectives are quadratic for the leader, linear for the first (*middle-level*) follower, and fractional for the second (*bottom-level*) follower. Non-linear objectives were added to IDs in [24] using non-linear optimal control approximations.

We model the problem as a MIXID with 3 agents, each with 2 objectives, and each variable observing the values of all previous variables. 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. We choose a utopian point for each agent and objective: $((0, 1000), (0, 100), (0, 1))$ and 3 pairs of weights:

$$((0.99999, 0.00001), (0.999, 0.001), (0.99, 0.01))$$

In repeated runs MIXID1 finds the correct policy $(0, 0, 9, 4, 1, 6, 0, 0)$ with objectives $((0, 396), (0, 21), (0, 0.2))$ with approximately 50% success.[1] The suboptimal solutions that it sometimes finds can be recognised by their lower z_1 values:
$((0, 196), (0, 17), (0, -0.06))$, $((0, 240), (0, 18), (0, 0))$ and $((0, 288), (0, 19), (0, 0.06))$.

3.7 Case study

As a case study to illustrate the system's ease of use, we now show the code required for our most complex example: the tri-level program of Section 2.7. MIXID1 is implemented in C, and the user must provide: (i) a text file specifying the variables, their agent numbers, their domains, their parent variables, how many objectives each agent has, and the conditional probability tables; (ii) a C function that takes as input a 1-dimensional array of (decision and chance) variable values, and computes as output a 2-dimensional array of agent-utility values.

Firstly, the text file describing the problem is shown in Figure 5 (the format is currently a simple research prototype but it could be adapted to a more user-friendly form). The meaning of the numbers is as follows:

```
8
1 6 8
2 3 0 8
3 16 0 1 8
3 27 0 1 2 8
3 20 0 1 2 3 8
3 31 0 1 2 3 4 8
3 16 0 1 2 3 4 5 8
3 21 0 1 2 3 4 5 6 8
8
8
2 0.0 1000.0 0.99999 0.00001
2 0.0 100.0 0.999 0.001
2 0.0 1.0 0.99 0.01
```

Fig. 5. Text file describing the tri-level MIXID

- The first line simply states that there are 8 variables numbered 0–7. We can therefore use "8" as a list terminator in the rest of the file.

[1] The first objective is given as 612 in [29] but it can be verified that the solution yields $z_1 = 396$.

– The next 8 rows describe each variable in turn: which agent it belongs to (0 would indicate a random variable but there are none in this example), its domain size D (values are assumed to be $0, 1, \ldots, D-1$), then a list of parent variables ending with terminator 8. As the highest agent number was 3, the method now knows that there are 3 agents. In this example each variable has all ancestors as parents.
– Next, each conditional probability table is listed, followed by a terminator. In this example there are no tables.
– Next, the user may provide a list of observed chance variable values, followed by a terminator. This feature is not used in this paper, but we briefly discuss it in Section 4.
– Finally, for each agent we input a number ω of objectives (2 in this example) followed by a list of ω weights, and a list of ω utopian values.

Secondly, we show the C `utility` function that computes the objectives. Its input is a 1-dimensional integer array called `values` and its output is a 2-dimensional real array called `results` (we assume that there will be no more than 10 agents and objectives, but these numbers can easily be increased):

```
void utility(int* values, float results[10][10]) {
```

To adapt the notation to that in Figure 4, and to adjust the variable domains to the desired ranges, we create new integer variables `x1`–`x8`:

```
int x1=values[0],   x2=values[1],
    x3=values[2]+5, x4=values[3]+4,
    x5=values[4]+1, x6=values[5],
    x7=values[6],   x8=values[7];
```

Compute the number of constraint violations:

```
int violations= !(-3*x1+7*x2+x3+x5==10)+
                !(14*x1+4*x2+x6==6)+
                !(x1+x2+x3-x4+x7==5)+
                !(2*x1+x2+2*x4+x8==8);
```

This becomes the first objective (index 0) for each agent:

```
results[1][0]=violations;
results[2][0]=violations;
results[3][0]=violations;
```

The other objective (index 1) for each agent is its given objective function z:

```
results[1][1]=(x1+x2+2*x3+4)*(-x1-x2+x3+2*x4+1);
results[2][1]=2*x2+x3+3*x4;
results[3][1]=((float) 2*x1+3*x2+2*x3-3*x4)/
              ((float) 5*x1+11*x2+x5+29); }
```

Note that no algorithmic details are required.

4 Conclusion

This paper makes two contributions. Firstly, it combines several known (and some new) ID extensions into a single framework we call a MIXID. Secondly, it proposes a MIXID solution method called MIXID1 based on RL, which has been proposed as a unifying approach to sequential decision making under uncertainty [31]. MIXID1 is a lightweight solver that does not need a graphics processing unit or other specialised hardware. We demonstrated the flexibility of the approach using examples taken from diverse literatures: ID, LIMID, multi-objective ID, stochastic programming, chance-constrained programming and tri-level programming. To the best of our knowledge, no other solver can tackle this range of optimisation problems, and we expect MIXIDs to be useful for hybrid optimisation problems that do not fit well into any particular class. A fruitful source of applications might be Safe RL in which constraints are used to ensure robust and safe policies.

Several methods exist for solving IDs. Some are exact and based on variable elimination [7, 22, 33, 34, 37] while others are approximate [4, 6, 27, 38, 39]. However, we know of only one other application of RL to solving IDs [32], 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 [10].

An aspect of IDs that we have ignored in this paper is *evidence propagation*. As in Bayesian networks, an ID might be provided with *evidence*: observed values of chance variables. This evidence can be propagated back to earlier chance variables to learn posterior distributions. Rejection sampling was used in [32] and also works with MIXID1, but this becomes impractical when probabilities are very small. Initial experiments with Gibbs sampling are promising, and evidence propagation will be investigated in future work. We are also interested in adding continuous variables, and using more powerful state aggregation than random tile coding: both improvements might be achieved by moving from SARSA to an actor-critic RL algorithm.

Acknowledgments

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.

References

1. S. Ahmed, M. Tawarmalani, N. V. Sahinidis. A Finite Branch-and-Bound Algorithm for Two-Stage Stochastic Integer Programs. *Mathematical Programming* **100**:355–377, 2004.
2. R. Arora, S. R. Arora. An Algorithm for Non-Linear Multi-Level Integer Programming Problems. *International Journal of Computing Science and Mathematics* **3**(3):211–225, 2010.

3. J. R. Birge, and F. V. Louveaux. Introduction to Stochastic Programming. Springer, 2011.
4. A. Cano, M. Gómez, S. Moral. A Forward-Backward Monte Carlo Method for Solving Influence Diagrams. *International Journal of Approximate Reasoning* **42**:119–135, 2006.
5. A. Charnes, W. W. Cooper. Chance-Constrained Programming. *Management Science* **6**(1):73–79, 1959.
6. J. M. Charnes, P. P. Shenoy. Multistage Monte Carlo Method for Solving Influence Diagrams Using Local Computation. *Management Science* **50**(3):405–418, 2004.
7. R. Dechter. A New Perspective on Algorithms for Optimizing Policies Under Uncertainty. In *Artificial Intelligence Planning Systems*, 2000, pp. 72–81.
8. M. Diehl, Y. Haimes. Influence Diagrams With Multiple Objectives and Tradeoff Analysis. *IEEE Transactions On Systems, Man, and Cybernetics Part A* 34(3):293–304, 2004.
9. M. M. K. Elshafei, M. S. El-Sherberry. Interactive Bi-level Multiobjective Stochastic Integer Linear Programming Problem. *Trends in Applied Sciences Research* **3**(2):154–164, 2008.
10. T. Everitt, R. Kumar, V. Krakovna, S. Legg. Modeling AGI Safety Frameworks with Causal Influence Diagrams. *Proceedings of the Workshop on Artificial Intelligence Safety, CEUR Workshop* vol. 2419, 2019.
11. Z. Gábor, Z. Kalmár, C. Szepesvári. Multi-criteria Reinforcement Learning. *Proceedings of the 15th International Conference on Machine Learning*, 1998, pp. 197–205.
12. Y. Gal, A. Pfeffer. Networks of Influence Diagrams: A Formalism for Representing Agents' Beliefs and Decision-Making Processes. *J. Artif. Intell. Res.* **33**:109–147, 2008.
13. J. García, F. Fernández. A Comprehensive Survey on Safe Reinforcement Learning. *Journal of Machine Learning Research* **16**:1437–1480, 2015.
14. I. Giagkiozis, P. J. Fleming. Methods for Multi-Objective Optimization: An Analysis. *Information Sciences* **293**:1–16, 2015.
15. K. Polich, G. Gmytrasiewicz. Interactive Dynamic Influence Diagrams. *Proceedings of the 6th International Joint Conference on Autonomous Agents and Multiagent Systems, Communications in Computer and Information Science* vol. 288, 2007, pp. 623–630.
16. J. González-Ortega, D. R. Insua, J. Cano. Adversarial Risk Analysis for Bi-agent Influence Diagrams: An Algorithmic Approach. *European Journal of Operational Research* **273**(3):1085–1096, 2019.
17. Shangding Gu, Long Yang, Yali Du, Guang Chen, Florian Walter, Jun Wang, Yaodong Yang, Alois C. Knoll: A Review of Safe Reinforcement Learning: Methods, Theory and Applications. CoRR abs/2205.10330 (2022)
18. R. A. Howard, J. E. Matheson. Influence Diagrams. Readings in Decision Analysis, Strategic Decisions Group, Menlo Park, CA, chapter 38, 1981, pp. 763–771.
19. S. H. Huang, A. Abdolmaleki, G. Vezzani, P. Brakel, D. J. Mankowitz, M. Neunert, S. Bohez, Y. Tassa, N. Heess, M. A. Riedmiller, R. Hadsell. A Constrained Multi-Objective Reinforcement Learning Framework. CoRL 2021:883–893.
20. The Effect of Hash Signature Collisions in a Chess Program. R. M. Hyatt, A. Cozzie. *ICGA Journal* **28**(3):131–139, 2005.
21. A. Jenzarli. Information/Relevance Influence Diagrams. In: *Proceedings of the 11th conference on Uncertainty in Artificial Intelligence (UAI)*, Quebec, Canada, pp. 329–337, 1995.
22. F. Jensen, V. Jensen, S. Dittmer. From Influence Diagrams to Junction Trees. In *Uncertainty in Artificial Intelligence*, 1994, pp. 367–363.
23. D. Koller, B. Milch. Multi-Agent Influence Diagrams for Representing and Solving Games. *Games and Economic Behavior* **45**(1):181–221, 2001.
24. V. Kratochvíl, J. Vomlel. Influence Diagrams for Speed Profile Optimization. *International Journal of Approximate Reasoning* **88**:567–586, 2017.
25. S. L. Lauritzen, D. Nilsson. Representing and Solving Decision Problems With Limited Information. *Management Science* **47**:1238–1251, 2001.

26. J. Lee, R. Marinescu, A. Ihler, R. Dechter. A Weighted Mini-Bucket Bound for Solving Influence Diagrams. *Proceedings of the Conference on Uncertainty in Artificial Intelligence*, 2019.
27. R. Marinescu, J. Lee, R. Dechter. A New Bounding Scheme for Influence Diagrams. *Proceedings of the 35th Conference on Artificial Intelligence*, 2021, pp. 12158–12165.
28. R. Marinescu, A. Razak, N. Wilson. Multi-objective Influence Diagrams. *Proceedings of the Conference on Uncertainty in Artificial Intelligence*, 2012.
29. S. Mishra, A. B. Verma. A Non-Differential Approach for Solving Tri-Level Programming Problems. *American International Journal of Research in Science, Technology, Engineering & Mathematics*, 2015.
30. K. van Moffaert, M. M. Drugan, A. Nowé. Scalarized Multi-Objective Reinforcement Learning: Novel Design Techniques. *Proceedings of the IEEE Symposium on Adaptive Dynamic Programming and Reinforcement Learning*, IEEE, 2013, pp. 191—199.
31. W. B. Powell. Reinforcement Learning and Stochastic Optimization: A Unified Framework for Sequential Decisions. Wiley, 2022.
32. S. D. Prestwich, F. Toffano, N. Wilson. A Probabilistic Programming Language for Influence Diagrams. *Proceedings of the 11th International Conference on Scalable Uncertainty Management*, 2017.
33. R. D. Shachter. Evaluating Influence Diagrams. *Oper. Res.* **34**(6):871–882, 1986.
34. P. Shenoy. Valuation-Based Systems for Bayesian Decision Analysis. *Operations Research* **40**(1):463–484, 1992.
35. J. Skalse, L. Hammond, C. Griffin, A. Abate. Lexicographic Multi-Objective Reinforcement Learning. *Proceedings of the 31st International Joint Conference on Artificial Intelligence*, 2022, pp. 3430–3436.
36. R. S. Sutton, A. G. Barto. Reinforcement Learning: an Introduction. MIT Press, Cambridge, MA, 1998.
37. Dynamic Programming and Influence Diagrams. *IEEE Transactions on Systems, Man, and Cybernetics* **20**(1):365–379, 1990.
38. W. Watthayu. Representing and Solving Influence Diagram in Multi-Criteria Decision Making: A Loopy Belief Propagation Method. *Proceedings of the International Symposium on Computer Science and its Applications*, 2008, pp. 118–125.
39. C. Yuan, X. Wu. Solving Influence Diagrams Using Heuristic Search. *Proceedings of the International Symposium on Artificial Intelligence and Mathematics*, 2010.
40. L. H. Zhou, L. Kevin, W. Y. Liu. Game theory-based Influence Diagrams. *Expert Systems* **30**(4):341–351, 2013.
41. A. L. Zobrist. A New Hashing Method with Application for Game Playing. Technical Report 88, Computer Sciences Department, University of Wisconsin, Madison, Wisconsin, 1969. Also: *International Computer Chess Association Journal* 13(2):69–73, 1990.