Solving influence diagrams: efficient mixed-integer programming formulation and heuristic

Helmi Hankimaa[1], Olli Herrala[1], Fabricio Oliveira[1], and Jaan Tollander de Balsch[1,2]

[1] Department of Mathematics and Systems Analysis, School of Science, Aalto University, Espoo, Finland
fabricio.oliveira@aalto.fi
[2] CSC – IT Center for Science, Espoo, Finland

Abstract. We propose novel mixed-integer linear programming (MIP) formulations to model decision problems posed as influence diagrams. We also present a novel heuristic that can be employed to warm start the MIP solver and provide heuristic solutions to more computationally challenging problems. We provide computational results showcasing the superior performance of these improved formulations as well as the performance of the proposed heuristic. Lastly, we describe a novel case study showcasing decision programming as an alternative framework for modelling multi-stage stochastic dynamic programming problems.

Keywords: decision problems under uncertainty · influence diagrams · decision analysis · mixed-integer programming.

1 Introduction

Influence diagrams [6] provide both a formal description of a decision problem and serve as a communication tool that requires minimal technical proficiency. Furthermore, they are useful in conveying structural relationships of the problem straightforwardly and thus crucially bridge the gap between quantitative specifications and qualitative descriptions. Due to their generality, influence diagrams pervade many modelling-based approaches that require a formal description of relationships between uncertainty, decisions and consequences.

Figure 1 shows an influence diagram for the N-monitoring problem, where N agents (A_1 to A_N) must decide whether to countervail an unknown load (L) based on imprecise readings of this load from their respective sensors (R_1 to R_N). The chance of failure (F) is influenced by the unknown load and the eventual decision to countervail the load. The final utility (T) is calculated considering whether the agents intervened and if a failure was observed. As such, this setting represents independent agents who must make decisions based only on observations of the state of the world but without being able to completely know it or share information among themselves.

Differently from decision trees, the nodes in an influence diagram do not need to be totally ordered, nor do they have to depend directly on all predecessors.

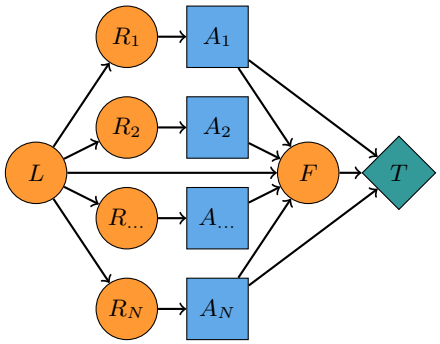


Fig. 1: Influence diagram for the N-monitoring problem [13]. Decisions are represented by squares, chance events by circles and consequences by diamonds

This freedom from dependence on all predecessors allows for the decisions to be made by, e.g., decision-makers who partially observe a common state of information (node L in Figure 1) but may differ in their ability to observe or are incapable of sharing information.

Unfortunately, quantitative methods employed to obtain optimal decisions from influence diagram representations typically require that some of that generality is curbed. Influence diagrams are, albeit more general, closely related to (possibly partially observable) Markov decision processes [11]. Thus, if (i) a single decision-maker is assumed (implying a total ordering among decision nodes) and (ii) the *no-forgetting* assumption holds (implying that each decision node and its direct predecessors influence all successor decision nodes), then the Markovian assumptions hold. This, in turn, enables one to solve influence diagrams with well-established techniques [2, 14]. A range of issues regarding the computational performance of solving influence diagrams are discussed in [1].

Many problems, including that in Figure 1, violate assumptions (i) and (ii). Indeed, there may be no memory or communication between deciding agents (meaning that they cannot know each other's decisions) or constraints imposed across the diagram, such as budget limitations or constraints representing risk (e.g., chance constraints [4] or conditional value-at-risk [12]). All of these either violate the assumption that the previous state is "remembered" at a later stage or that all information influencing the decision alternatives is known when making said decision.

In [9], the authors proposed an analytical framework to characterise *limited memory influence diagrams*. Note that the notion of limited memory can also be used to encompass settings with multiple decision-makers, the limited or absent sharing of information being its defining feature (regardless of whether it is due to lack of memory or communication between decision-makers). In any case, limited-memory influence diagrams are essentially influence diagrams that do

not satisfy assumptions (i) and (ii). However, the fact that they do not satisfy assumptions (i) and (ii) means that they require more sophisticated methods such as branch-and-bound [8] that can extract optimal decisions from these diagrams.

Aiming to address the aforementioned limitations, decision programming [13] leverages the capabilities of stochastic programming and decision analysis to model and solve multi-stage decision problems using mathematical optimisation techniques. In essence, decision programming exploits the expressiveness of (limited memory) influence diagrams in structuring problems to develop deterministic equivalent [3] mixed-integer programming (MIP) formulations.

In this paper, we provide multiple contributions in terms of practical and methodological aspects associated with decision programming. Specifically, we present a novel formulation for decision programming problems that is considerably more efficient from a computational standpoint. Furthermore, we propose a heuristic inspired by the single policy update heuristic, originally proposed in [9], that can efficiently generate good feasible solutions for the computationally challenging problems considered in our experiments.

Finally, we present a case study in which we use decision programming to develop an optimal decision strategy for allocating preventive care for coronary heart disease (CHD). This study aims to evaluate the suitability of decision programming for performing the cost-benefit analysis originally performed by [7]. In [7], a set of alternative predefined testing and treatment strategies for CHD are optimised using dynamic programming. We show how the same problem can be solved precluding the need to define strategies a priori.

This paper is structured as follows. Section 2 presents the technical details associated with the decision programming framework. In Section 3, we present the novel formulation proposed in this paper, followed by the proposed adaptation of the single policy update heuristic presented in Section 4.

In Section 5, we provide computational results showcasing the benefits of the methodological innovations and in Section 6 we describe the case study considering optimal preventive care strategies for CHD. Lastly, in Section 7 we provide conclusions.

2 Decision Programming

An influence diagram can be defined as an acyclic graph $G(N, A)$ formed by nodes in $N = C \cup D \cup V$, where C is a subset of chance nodes, D a subset of decision nodes, and V a subset of value nodes. Value nodes represent consequences incurred from decisions made at nodes D and chance events observed at nodes C. Each decision and chance node $j \in C \cup D$ can assume a state s_j from a discrete and finite set of states S_j. For a decision node $j \in D$, S_j represents the decision alternatives. For a chance node $j \in C$, S_j is the set of possible outcomes.

In the diagram, arcs represent interdependency among decisions and chance events. Set $A = \{(i, j) \mid i, j \in N\}$ contains the arcs (i, j), which represent the influence between nodes i and j. This influence is propagated in the diagram

in the form of *information*. That is, an arc (i, j) that points to a decision node $j \in D$ indicates that the decision at $j \in D$ is made *knowing* the realisation (i.e., uncertainty outcome or decision made) of state $s_i \in S_i$, with $i \in C \cup D$; an arc that points to a chance node $j \in C$ indicates that the realisation $s_j \in S_j$ is dependent (or conditional) on realisation $s_i \in S_i$ of node $i \in C \cup D$.

The *information set* $I(j) = \{i \in N \mid (i, j) \in A\}$ comprises all immediate predecessors (or parents) of a given node $j \in N$. Despite being a less common terminology, we opt for the term "information set" to highlight the role of information in the modelling of the decision process. The decisions $s_j \in S_j$ made in each decision node $j \in D$ depend on their *information state* $s_{I(j)} \in S_{I(j)}$, where $S_{I(j)} = \prod_{i \in I(j)} S_i$ is the set of all possible information states for node j. Analogously, the possible realisations $s_j \in S_j$ for each chance node $j \in C$ and their associated probabilities also depend on their information state $s_{I(j)} \in S_{I(j)}$.

Let us define $X_j \in S_j$ as the realised state at a chance node $j \in C$. For a decision node $j \in D$, let $Z_j : S_{I(j)} \to S_j$ be a mapping between each information state $s_{I(j)} \in S_{I(j)}$ and decision $s_j \in S_j$. That is, $Z_j(s_{I(j)})$ defines a local decision strategy, which represents the choice of some $s_j \in S_j$ in $j \in D$, given the information $s_{I(j)}$. Such a mapping can be represented by an indicator function $\mathbb{I} : S_{I(j)} \times S_j \to \{0, 1\}$ defined so that

$$\mathbb{I}(s_{I(j)}, s_j) = \begin{cases} 1, & \text{if } Z_j \text{ maps } s_{I(j)} \text{ to } s_j, \text{ i.e., } Z_j(s_{I(j)}) = s_j; \\ 0, & \text{otherwise.} \end{cases}$$

A (global) *decision strategy* is the collection of local decision strategies in all decision nodes: $Z = (Z_j)_{j \in D}$, selected from the set of all possible strategies $\mathbb{Z}$.

A *path* is a sequence of states $s = (s_i)_{i=1,\ldots,n}$, with $n = |C| + |D|$ and

$$S = \{(s_i)_{i=1,\ldots,n} \mid s_i \in S_i, i = 1, \ldots, n\} \tag{1}$$

is the set of all possible paths. We assume that the nodes $C \cup D$ are numbered from 1 to n such that for each arc $(i, j) \in A$, $i < j$. Moreover, we say that a strategy Z is compatible with a path $s \in S$ if $Z_j(s_{I(j)}) = s_j$ for all $j \in D$. We denote as $S(Z) \subseteq S$ the subset of all paths that are compatible with a strategy Z.

Using the notion of information states, the conditional probability of observing a given state s_j for $j \in C$ is $\mathbb{P}(X_j = s_j \mid X_{I(j)} = s_{I(j)})$. The probability associated with a path $s \in S$ being observed given a strategy Z can then be expressed as

$$\mathbb{P}(s \mid Z) = \left(\prod_{j \in C} \mathbb{P}(X_j = s_j \mid X_{I(j)} = s_{I(j)}) \right) \left(\prod_{j \in D} \mathbb{I}(s_{I(j)}, s_j) \right). \tag{2}$$

Notice that the term $\prod_{j \in D} \mathbb{I}(s_{I(j)}, s_j)$ in equation (2) takes value one if the strategy Z is compatible with the path $s \in S$, being zero otherwise. Furthermore,

notice that one can pre-calculate the probability

$$p(s) = \left(\prod_{j \in C} \mathbb{P}(X_j = s_j \mid X_{I(j)} = s_{I(j)}) \right) \tag{3}$$

of a path $s \in S$ being observed, in case a compatible strategy is chosen.

At the value node $v \in V$, a real-valued utility function $U_v : S_{I(v)} \to \mathbb{R}$ maps the information state $s_{I(v)}$ to a utility value $U_v(s_{I(v)})$. We usually assume the utility value of a path s to be the sum of individual value nodes' utilities: $U(s) = \sum_{v \in V} U_v(s_{I(v)})$. The default objective is to choose a strategy $Z \in \mathbb{Z}$ maximising the expected utility, which can be expressed as

$$\max_{Z \in \mathbb{Z}} \sum_{s \in S} \mathbb{P}(s \mid Z) U(s). \tag{4}$$

Notice that other objective functions can also be modelled. For example, in [13], the authors discuss the use of the conditional value-at-risk.

To formulate this into a mathematical optimisation problem, we start by representing the local strategies Z_j using binary variables $z(s_j \mid s_{I(j)})$ that take value one if $\mathbb{I}(s_{I(j)}, s_j) = 1$, and 0 otherwise. We then observe that using (2) and (3), the objective function (4) becomes

$$\max_{z} \sum_{s \in S} p(s) U(s) \prod_{j \in D} z(s_j \mid s_{I(j)}).$$

This function is nonlinear and is used only for illustrating the nature of the formulations. The usefulness of this construction becomes more obvious in Section 3. In [13] the conditional path probability $\mathbb{P}(s \mid Z)$ in (4) is instead replaced with a continuous decision variable $\pi(s)$, enforcing the correct behaviour of this variable using affine constraints.

With these building blocks, the problem can be formulated as a mixed-integer linear programming (MILP) model, which allows for employing off-the-shelf mathematical programming solvers. The MILP problem presented in [13] can be stated as (5)-(10).

$$\max_{Z \in \mathbb{Z}} \sum_{s \in S} \pi(s) U(s) \tag{5}$$

$$\text{subject to} \sum_{s_j \in S_j} z(s_j \mid s_{I(j)}) = 1, \qquad \forall j \in D, s_{I(j)} \in S_{I(j)}, \tag{6}$$

$$0 \le \pi(s) \le p(s), \qquad \forall s \in S, \tag{7}$$

$$\pi(s) \le z(s_j \mid s_{I(j)}), \qquad \forall j \in D, s \in S, \tag{8}$$

$$\pi(s) \ge p(s) + \sum_{j \in D} z(s_j \mid s_{I(j)}) - |D|, \quad \forall s \in S, \tag{9}$$

$$z(s_j \mid s_{I(j)}) \in \{0, 1\}, \qquad \forall j \in D, s_j \in S_j, s_{I(j)} \in S_{I(j)}. \tag{10}$$

Variables $\pi(s)$ are nonnegative continuous variables representing the conditional path probability in equation (2). They take the value of the path probability $p(s)$ in case the selected strategy Z is compatible with the path $s \in S$ and zero otherwise. Notice that this compatibility is equivalent to observing $z(s_j \mid s_{I(j)}) = 1$ for all $s_j \in S$ such that $j \in D$.

The objective function (5) defines the expected utility value, calculated considering only the paths that are compatible with the strategy. Constraint (6) enforces the one-to-one nature of the mapping $\mathbb{I}(s_{I(j)}, s_j)$, represented by the z-variables. The correct behaviour of variables $\pi(s)$ is guaranteed by constraints (7)-(9), which enforce that $\pi(s) = p(s)$ if $z(s_j \mid s_{I(j)}) = 1$ for all $s_j \in S$ such that $j \in D$. The term $|D|$ in (9) represents the cardinality of the set D. Notice that (7) defines the domain of $\pi(s)$.

3 Improved formulations

Next, we present reformulations developed to enhance the numerical performance of the formulation (5)–(10). For that, let us first define the subset of paths

$$S_{s_j|s_{I(j)}} = \left\{ s \in S \mid (s_{I(j)}, s_j) \subseteq s \right\}.$$

Notice that we use the notation $(s_{I(j)}, s_j)$ to represent a portion of a path s, formed by the combination of the information state $s_{I(j)}$ (which may itself be a collection of states, if $|I(j)| > 1$) and the state s_j. We also utilise the set operator $\subseteq$ to indicate that the states $(s_{I(j)}, s_j)$ are part of the path $s \in S$. Notice that the states $(s_{I(j)}, s_j)$ do not need to be consecutive in the path s, although the ordering between $s_{I(j)}$ and s_j is naturally preserved in s.

Considering $j \in D$, the subset $S_{s_j|s_{I(j)}}$ allows us to define the notion of *locally compatible paths*, that is, the collection of paths s compatible with local strategies Z_j for which $\mathbb{I}(s_{I(j)}, s_j) = 1$. The definition of the subset $S_{s_j|s_{I(j)}}$ allows us to derive the following valid inequality for (5)–(10):

$$\sum_{s \in S_{s_j|s_{I(j)}}} \pi(s) \leq z(s_j \mid s_{I(j)}), \quad \forall j \in D, s_j \in S_j, s_{I(j)} \in S_{I(j)}. \tag{11}$$

Constraint (11) states that only paths that are compatible with the selected strategy might be allowed to have a probability different than zero. Moreover, since it is enforced on all decision nodes, it means that this constraint guarantees that only the paths that are compatible with the strategy Z are active. Recall that we denote this set of compatible paths as $S(Z) \subseteq S$.

As pointed out in [13], for expected utility maximisation, constraint (9), which prevents variables $\pi(s)$ from wrongly taking value zero, is only required when some of the utility values $U(s)$, $s \in S$, are negative. Notice that this is otherwise prevented by the maximisation of the objective function (5), naturally steering these variables to their upper bound values. Another way to guarantee that the variables $\pi(s)$ take their correct value, i.e., $\pi(s) = p(s)$, if $s \in S(Z)$, is

to impose the constraint

$$\sum_{s \in S} \pi(s) = 1. \tag{12}$$

As it will be discussed in Section 5, replacing (8) and (9) with (11) and (12) provides considerable gains in terms of linear relaxation strengthening. Furthermore, we observe that the computational performance can be even further improved by employing a simple variable substitution. Recall that in the original formulation (5)-(10), variables $\pi(s)$ represent the conditional path probability $\mathbb{P}(s \mid Z) = p(s) \prod_{j \in D} z(s_j \mid s_{I(j)})$. If we let $x(s) \in [0,1]$, $s \in S$ represent the product $\prod_{j \in D} z(s_j \mid s_{I(j)})$, then we can reformulate the problem by substituting $\pi(s) = p(s)x(s)$ for all $s \in S$.

Although $x(s)$, $\forall s \in S$, is continuous, it behaves as a binary variable that takes value one whenever the path is compatible with the strategy and zero, otherwise. This is analogous to the behaviour of variable $\pi(s) \in [0, p(s)]$ in (5)–(10). We highlight that, from a theoretical standpoint, there is no obvious reason for performing such a substitution. On the other hand, we will show in the computational experiments presented in Section 6 that it yields significant practical benefits in terms of computational performance.

Using these x-variables, we can reformulate (11) as

$$\sum_{s \in S_{s_j \mid s_{I(j)}}} x(s) \leq |S_{s_j \mid s_{I(j)}}| z(s_j \mid s_{I(j)}), \quad \forall j \in D, s_j \in S_j, s_{I(j)} \in S_{I(j)}, \tag{13}$$

a consequence of $x(s) \in [0,1]$ and the fact that $z(s_j \mid s_{I(j)})$ must be equal to 1 for $x(s)$ to be positive for $s \in S_{s_j \mid s_{I(j)}}$.

Constraint (13) can be strengthened further. We note that a path must be in the set of compatible paths $S(Z)$ for $x(s)$ to be positive with strategy Z. Using this information, we can infer a tighter upper bound for the number of paths that can be active ($x(s) > 0$) from the set of locally compatible paths. We observe that in a set of compatible paths $S(Z)$, each information state $s_{I(j)}$ maps to exactly one decision alternative s_j for each decision node $j \in D$, by constraint (6). However, the set of locally compatible paths for a given pair of information state and decision node state $(s_{I(j)}, s_j)$ of decision node $j \in D$, includes paths for all combinations $(s_{I(k)}, s_k)$ of information states and decisions for the other decision nodes $k \in D \setminus \{j\}$. Hence, only a fraction of the locally compatible paths can be active. The fraction is linked to the number of states $|S_k|$ of the other decision nodes $k \in D \setminus \{j\}$. The number of locally compatible paths that will also be active, i.e., $|S_{s_j \mid s_{I(j)}} \cap S(Z)|$ can be defined as

$$|S_{s_j \mid s_{I(j)}} \cap S(Z)| = \frac{|S_{s_j \mid s_I(j)}|}{\Pi_{k \in D \setminus (\{j\} \cup I(j))} |S_k|}. \tag{14}$$

Notice that the calculation of the number of active paths must consider that some decision nodes may be part of the information state $I(j)$ of node $j \in D$, and, as such, will have their states observed (or fixed) in the set $S_{s_j \mid s_I(j)}$. Therefore,

these decision nodes must be excluded from the product in the denominator in (14). Using (14), we can reformulate (13) into the strengthened form

$$\sum_{s \in S_{s_j|s_{I(j)}}} x(s) \leq \frac{|S_{s_j|s_I(j)}|}{\Pi_{k \in D \setminus (\{j\} \cup I(j))} |S_k|} z(s_j \mid s_{I(j)}), \quad \forall j \in D, s_j \in S_j, s_{I(j)} \in S_{I(j)}. \tag{15}$$

One last aspect that can be taken into account is that, depending on the problem structure, some sequence of states $s = (s_i)_{i=1,\ldots,n}$ forming a path may never be observed and can be preemptively filtered out from the set of paths S. This is the case, for example, in problems where earlier decisions or uncertain events dictate whether alternatives or uncertainties are observed. To prevent the assembling of these unnecessary paths, we consider a set of *forbidden* paths, which, once removed, lead to a set $S^* \subseteq S$ of *effective* paths. Notice that these forbidden paths have probability zero by the structure of the problem, and therefore their removal does not affect the expected utility nor the constraints of the model but allows for significant savings in terms of the scale of the model.

One issue emerges in settings where $S^* \subset S$ regarding the term (14). Notice that the bound is based on the premise that we can infer the total number of paths by considering the Cartesian product of the state sets S_j, $j \in N$. However, as forbidden paths are removed, some of the x-variables corresponding to paths $s \in S_{s_j|s_I(j)}$ might be removed, making inequality (15) loose. A simple safeguard for this is to consider

$$\Gamma(s_j|s_{I(j)}) = \min \left\{ |S^*_{s_j|s_{I(j)}}|, \frac{|S_{s_j|s_I(j)}|}{\Pi_{k \in D \setminus (\{j\} \cup I(j))} |S_k|} \right\} \tag{16}$$

and reformulate (15) as

$$\sum_{s \in S_{s_j|s_{I(j)}}} x(s) \leq \Gamma(s_j|s_{I(j)}) z(s_j \mid s_{I(j)}), \quad \forall j \in D, s_j \in S_j, s_{I(j)} \in S_{I(j)}. \tag{17}$$

Combining the above, we can reformulate (5)–(10) as follows.

$$\underset{Z \in \mathbb{Z}}{\text{maximise}} \sum_{s \in S^*} U(s) p(s) x(s) \tag{18}$$

$$\text{subject to} \sum_{s_j \in S_j} z(s_j \mid s_{I(j)}) = 1, \qquad \forall j \in D, s_{I(j)} \in S_{I(j)} \tag{19}$$

$$\sum_{s \in S_{s_j|s_{I(j)}}} x(s) \leq \Gamma(s_j|s_{I(j)}) z(s_j \mid s_{I(j)}), \quad \forall j \in D, s_j \in S_j, s_{I(j)} \in S_{I(j)} \tag{20}$$

$$\sum_{s \in S^*} p(s) x(s) = 1, \tag{21}$$

$$0 \leq x(s) \leq 1, \qquad \forall s \in S^* \tag{22}$$

$$z(s_j \mid s_{I(j)}) \in \{0, 1\}, \qquad \forall j \in D, s_j \in S_j, s_{I(j)} \in S_{I(j)}. \tag{23}$$

where $\Gamma(s_j|s_{I(j)})$ is defined as in (16). Note that this formulation preserves the (mixed-integer) linear nature of (5)–(10).

4 Primal heuristic: single policy update (SPU)

A notable contribution of [9] is the single policy update (SPU) heuristic for obtaining "locally optimal" strategies in the sense that the corresponding solutions cannot be improved by changing only one of the local strategies $Z_j(s_{I(j)})$.

Our proposed heuristic is loosely based on the ideas in [9], as described in Algorithm 1. The first step of the heuristic is to obtain a random strategy Z (note that this too is a heuristic, albeit a very simple one). Additionally, we initialise the *lastImprovement* variable that will be used to stop the algorithm after finding a local optimum. The strategy Z is then iteratively improved by examining each information state $s_{I(j)} \in S_{I(j)}$ for each decision node $j \in D$ in order, choosing the local strategy $Z'_j(s_{I(j)})$ maximising the expected utility.

$Z \leftarrow randomstrategy()$;
$lastImprovement \leftarrow (undef, undef)$;
while *true* **do**
  for $j \in D$, $s_{I(j)} \in S_{I(j)}$ **do**
    if $(j, s_{I(j)}) = lastImprovement$ **then**
      return Z;
    else
      $Z'_j(s_{I(j)}) \leftarrow bestLocalStrategy(Z, j, s_{I(j)})$;
      $Z' \leftarrow modifyStrategy(Z, Z'_j(s_{I(j)}))$;
      if $EU(Z') > EU(Z)$ **then**
        $Z \leftarrow Z'$;
        $lastImprovement \leftarrow (j, s_{I(j)})$;
      end
    end
  end
  if $(undef, undef) = lastImprovement$ **then return** Z ;
end

Algorithm 1: The single policy update heuristic

This process of locally improving the strategy is performed repeatedly for all pairs $(j, s_{I(j)})$ until no improvement is made during a whole iteration through the set of such pairs, that is, $(j, s_{I(j)}) = lastImprovement$ or $(undef, undef) = lastImprovement$ if the initial random strategy cannot be improved. The number of possible strategies Z is finite, and the algorithm thus converges in a finite number of iterations. It is also easy to see that at termination, there is no possible local improvement and the strategy Z is thus, in that sense, locally optimal. In [9], the authors show that for *soluble* LIMIDs, this heuristic results in a globally optimal solution. However, influence diagrams are not generally soluble. The performance of the heuristic is explored in Section 5.

5 Computational experiments

We present a collection of computational experiments[3] carried out to assess the performance of the proposed reformulation (18)-(23) against the original formulation (5)-(10) presented in [13].

We also present computational results highlighting the performance of the proposed SPU heuristic. The problems used for testing are (i) the pig farm problem originally from [9] modified to allow for artificially augmenting the number of time periods and generating input parameters randomly; (ii) the N-monitoring problem, as proposed in [13], in which we also can artificially augment the number of decision agents and randomly generate instances. These two problems have a major difference in their structure: while the N-monitoring problem has N decisions in parallel with no communication between the decision-makers, the pig farm problem is a partially observed Markov decision process (POMDP) where decisions are made in series with limited memory of the past. This structural difference leads to the N-monitoring problem having a larger tree-width, which has generally been an issue for solving influence diagrams [10].

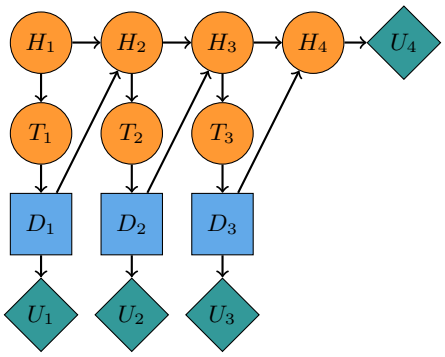
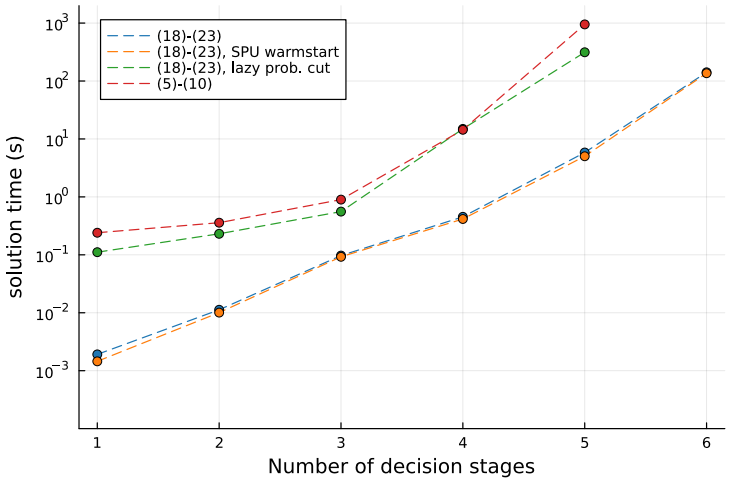


Fig. 2: The influence diagram of the pig farm problem [13].

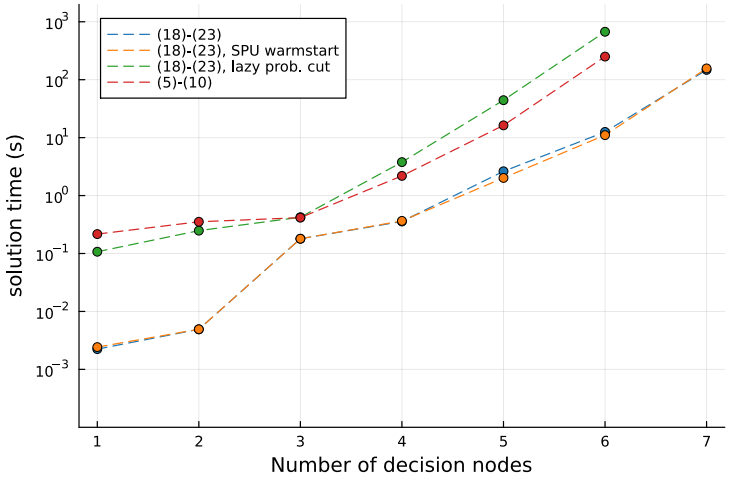
The pig farm problem is presented in Figure 2, where nodes H_i represent the health of a pig, nodes T_i represent the result of a diagnostic test, and nodes D_i correspond to treatment decisions. Treating a pig incurs a cost represented by the value nodes U_i for $i \in \{1, 2, 3\}$ and in the end, the pig can be sold for a price depending on the final health state. This problem was chosen for the computational comparison because similar low tree-width structures frequently

[3] All experiments are run using 16 GB of memory and 8 threads on an Intel Xeon Gold 6248 CPU and the code can be found in github.com/gamma-opt/DecisionProgramming.jl.

arise in contexts such as quality control [5] or testing and treating patients for a disease [7], discussed in Section 6 of this paper.



(a) The pig farm problem



(b) The N-monitoring problem

Fig. 3: The solution times of the two example problems with different numbers of decision nodes using different formulations. Notice the logarithmic y-axis.

Figure 3 shows the increase in average solution times over 50 instances as the number of decision stages increases in the two example problems. For the original formulation (5)-(10), [13] show that solution times are greatly improved

by adding a *probability cut* $\sum_{s \in S} \pi(s) = 1$ as a lazy constraint to the model. This approach is thus used in the computational experiments for the original formulation. For (18)-(23), a similar constraint is included in the formulation by default. However, we additionally analyse an instance of the reformulated model (18)-(23), where constraint (21) is implemented as a lazy constraint.

For both problems, the rate of increase in the solution times quickly renders the original formulation (5)-(10) computationally intractable, as seen in Fig 3. In contrast, the solution times for the improved formulation (18)-(23) are consistently at least one order of magnitude faster. Finally, the lazy probability cut that was found to improve solution times in [13] is detrimental to computational performance in the new formulation (18)-(23).

In Table 1, we present statistics on the quality of the LP relaxation. As discussed before, the hypothesis is that the formulation (18)-(23) is considerably tighter than (5)-(10). The results in Table 1 confirms this hypothesis as more than half of the LP relaxation solutions for the novel formulation are within 25% of the optimal solution, whilst the formulation (5)-(10) provides optimality gaps that are orders of magnitude larger.

	(5)-(10)	(18)-(23)
10th percentile	15.4	1.00
median	26.4	1.21
90th percentile	31.1	1.81
mean	25.0	1.34

Table 1: Statistics of the root relaxation quality relative to the optimal solution for 50 randomly generated pig farm problems with 5 decision stages. The solutions are scaled so that a value of 1 corresponds to the optimal solution.

Figure 4 shows the process of improving solutions in the single policy update (SPU) heuristic. For the 50 instances in this test set, the last solution is found within eight seconds, and the solution is the global optimum in all but one of the instances. Note that [9] showed that this version of the pig farm problem is not soluble, and thus the SPU heuristic is not guaranteed to find the optimal solution. We observe that while the single policy update heuristic is successful in finding good initial solutions quickly, the effect of providing the solver with these initial solutions is negligible (see Figure 3).

6 Case study: optimal preventive healthcare for CHD

In this case study we use decision programming to optimise the use of traditional and genetic testing to support the targeting of statin medication treatment for preventing coronary heart disease (CHD). This case study is replicated from [7], where the authors developed a testing and treatment strategy by optimising

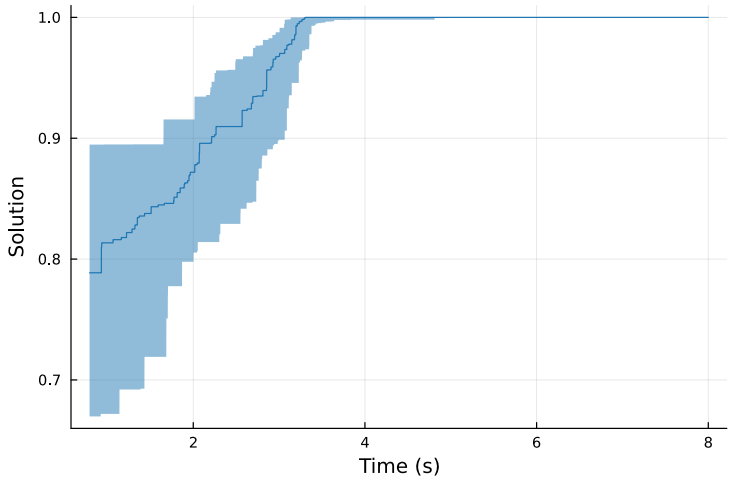


Fig. 4: The median and first and third quartiles of solutions found by the SPU heuristic in 50 randomly generated pig farm problems with 6 decision stages. The solutions are scaled so that a value of 1 corresponds to the optimal solution.

net monetary benefit (NMB), a cost-benefit objective consisting of the health outcomes and testing costs within a 10-year time horizon.

The problem setting is such that the patient is assumed to have a prior risk estimate R_0. A risk estimate is a prediction of the patient's chance of having a CHD event in the next ten years. The risk estimates are grouped into risk levels, which range from 0% to 100% with a suitable discretisation, e.g., $S_{R_0} = \{0\%, 1\%, ..., 99\%, 100\%\}$. The first testing decision T_1 is made based on the prior risk estimate. This entails deciding whether to perform traditional risk factors (TRS), genetic risk factors (GRS) or if no testing is needed. If a test is conducted, the risk estimate is updated (R_1) and based on the new information a second testing decision T_2 follows. It entails deciding whether further testing should be conducted or not. The second testing decision is constrained so that the same test that was conducted in the first stage cannot be repeated. If a second test is conducted, the risk estimate is updated again (R_2). The treatment decision T_D (whether the patient receives preventive statin medication or not) is made based on the resulting risk estimate of this testing process. Note that if no tests are conducted, the treatment decision is made based on the prior risk estimate. Figure 5 provides an influence diagram for the decision problem.

An interesting result is that the optimal strategy found by our model is the same strategy that was deemed the best among strategies tested in [7]. In a way, this provides optimality guarantees to their results, which, in principle, they could not have determined without exhaustively assessing all possible testing strategies. Figure 6 illustrates the strategy obtained by our model, indicating also the thresholds found in [7] for comparison.

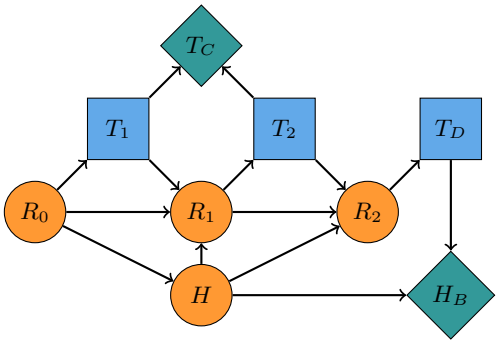


Fig. 5: Influence diagram for optimising a CHD preventive care decision strategy.

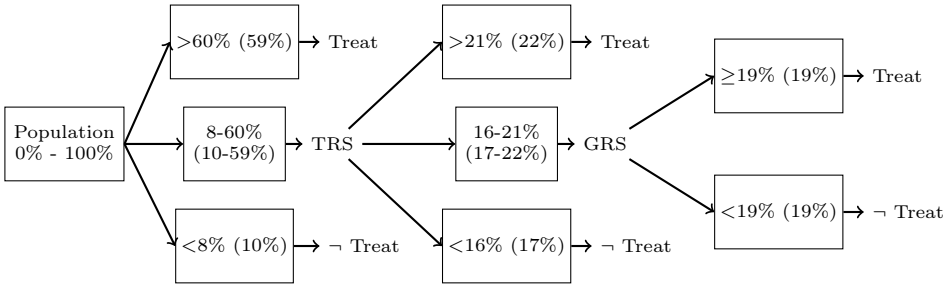


Fig. 6: Optimal strategy obtained by our model (in parentheses, the original value from [7]). Small differences are related to input rounding

Finally, we perform a computational assessment in line with Section 5. Due to the computational complexity of the model, the problem was decomposed into a 101 subproblems corresponding to the prior risk levels 0%, 1%, ..., 100%. To assess computational performance of the two formulations (18)-(23) and (5)-(10), we compare the solution times of the 101 subproblems. The results presented in Figure 7 show that the formulation (5)-(10) from [13] could not find the optimal solution within 2 hours for almost a third of the subproblems (31 of 101 indicated with orange). On the other hand, nearly all subproblems (98 of 101) were solved in under 10 minutes using the proposed formulation (18)-(23).

7 Conclusions

In this paper, we expand on the ideas originally proposed in [13] providing multiple methodological enhancements. These include a novel and more efficient

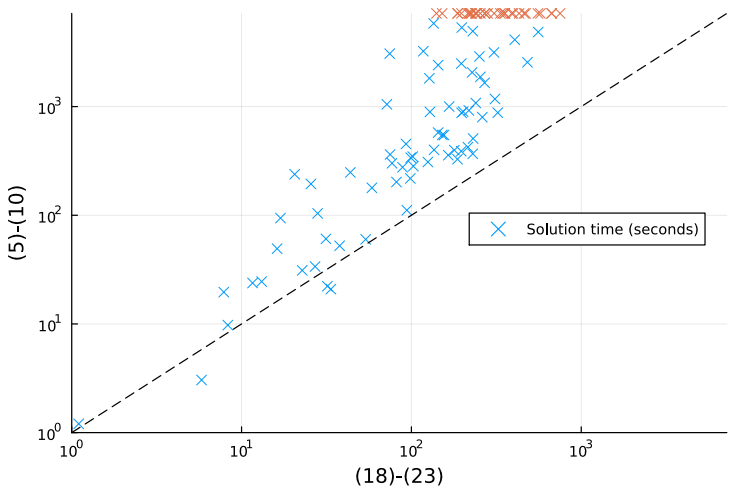


Fig. 7: The solution times for each subproblem (prior risk level) in the CHD prevention case study. Points above the dashed line correspond to the new formulation being faster; orange points correspond to termination due to time limit.

formulation, valid bounds to tighten relaxations, and a heuristic that can be used to find feasible solutions and, consequently, to warm start the MILP solver.

Furthermore, we conduct a case study based on the study originally proposed by [7]. The case study demonstrates that the proposed models can be used in settings that would normally require resorting to more ad-hoc computational tools, lending themselves to be a general and accessible tool for practitioners. We believe that this will allow for a much wider range of practitioners and researchers to have access to mathematical optimisation-based tools for supporting decision-making. Furthermore, this will create novel inroads for the use of mathematical optimisation in the area of decision analysis at large, potentially unveiling new and promising directions for future developments.

Acknowledgments. We are enormously grateful to Juho Andelmin, whose initial implementations led to the development of `DecisionProgramming.jl`. We are also grateful for the contributions of several students to the development of the package, as well as to the welcoming and supportive `JuMP` community. We acknowledge the financial support from the Research Council of Finland (decision number 332180) and the computer resources from the Aalto University School of Science "Science-IT" project.

Disclosure of Interests. The authors have no competing interests to declare that are relevant to the content of this article.

Bibliography

[1] Bielza, C., Gómez, M., Rıos-Insua, S., del Pozo, J.F.: Structural, elicitation and computational issues faced when solving complex decision making problems with influence diagrams. Computers & Operations Research **27**(7-8), 725–740 (2000)

[2] Bielza, C., Gómez, M., Shenoy, P.P.: A review of representation issues and modeling challenges with influence diagrams. Omega **39**(3), 227–241 (2011)

[3] Birge, J.R., Louveaux, F.: Introduction to stochastic programming. Springer Science & Business Media (2011)

[4] Charnes, A., Cooper, W.: Chance constrained programming. Management Science **6**(1), 73–79 (1959)

[5] Cobb, B.R.: Intermittent sampling for statistical process control with the number of defectives. Computers & Operations Research **161**, 106423 (2024)

[6] Howard, R.A., Matheson, J.E.: Influence diagrams. Decision Analysis **2**(3), 127–143 (2005)

[7] Hynninen, Y., Linna, M., Vilkkumaa, E.: Value of genetic testing in the prevention of coronary heart disease events. PloS one **14**(1), e0210010 (2019)

[8] Khaled, A., Hansen, E.A., Yuan, C.: Solving limited-memory influence diagrams using branch-and-bound search. In: Uncertainty in Artificial Intelligence (UAI-13). pp. 331–341. AUAI Press, Arlington, Virginia, USA (2013)

[9] Lauritzen, S.L., Nilsson, D.: Representing and solving decision problems with limited information. Management Science **47**(9), 1235–1251 (2001)

[10] Mauá, D.D., de Campos, C.P., Zaffalon, M.: Solving limited memory influence diagrams. Journal of Artificial Intelligence Research **44**, 97–140 (2012)

[11] Puterman, M.L.: Markov decision processes. Handbooks in operations research and management science **2**, 331–434 (1990)

[12] Rockafellar, R., Uryasev, S.: Optimization of conditional value-at-risk. Journal of Risk **2**, 21–42 (2000)

[13] Salo, A., Andelmin, J., Oliveira, F.: Decision programming for mixed-integer multi-stage optimization under uncertainty. European Journal of Operational Research **299**(2), 550–565 (2022)

[14] Shachter, R.D., Bhattacharjya, D.: Solving influence diagrams: Exact algorithms. Wiley encyclopedia of operations research and management science (2010)