An Algorithm for Generating Most Probable Paths in Decision Programming

Jerry Aunula

School of Science

Bachelor's thesis
Espoo 27.8.2021

Supervisor

Prof. Ahti Salo

Advisor

M.Sc. (Tech.) Juho Roponen

Aalto University
School of Science

Aalto University
School of Science

Aalto University, P.O. BOX 11000, 00076 AALTO
www.aalto.fi
Abstract of the bachelor's thesis

Author Jerry Aunula

Title An Algorithm for Generating Most Probable Paths in Decision Programming

Degree programme Engineering Physics and Mathematics

Major Mathematics and Systems Sciences **Code of major** SCI3029

Teacher in charge Prof. Ahti Salo

Advisor M.Sc. (Tech.) Juho Roponen

Date 27.8.2021 **Number of pages** 18+1 **Language** English

Abstract

Influence diagrams visually represent decision problems. Several methods for solving these diagrams with multiple decision stages and endo- and exogenous uncertainty have been presented in the literature. However, exact approaches for Limited Memory Influence Diagrams (LIMID) and influence diagrams with problem-spanning constraints are scarce. The Decision Programming framework is a general method for solving influence diagrams, which works by converting the problem to a Mixed Integer Linear Programming form, solvable by off-the-shelf solvers. However, Decision Programming does not scale up for large problems, which limits its applicability. This thesis develops an algorithm that produces a sub-problem of the original problem to produce an approximate solution more quickly. We tested the algorithm's performance on Prognostics and Health Management problem concerning the maintenance optimization of a sensor-turbine system. We found that our algorithm managed this problem quite well, reducing the problem size by 95% while losing less than 0.5% of the expected utility.

Keywords Influence diagram, Decision Programming, stochastic programming, approximation algorithm

Aalto University
School of Science

Aalto-yliopisto, PL 11000, 00076 AALTO
www.aalto.fi
Tekniikan kandidaatintyön tiivistelmä

Tekijä Jerry Aunula

Työn nimi Algoritmi todennäköisimpien polkujen generoimiseen Decision Programming -optimointiviitekehykselle

Koulutusohjelma Teknillinen fysiikka ja matematiikka

Pääaine Matematiikka ja systeemitieteet **Pääaineen koodi** SCI3029

Vastuuopettaja Prof. Ahti Salo

Työn ohjaaja DI Juho Roponen

Päivämäärä 27.8.2021 **Sivumäärä** 18+1 **Kieli** Englanti

Tiivistelmä

Vaikutuskaaviot ovat visuaalinen tapa esittää päätösongelmia. Kirjallisuudessa on esitetty useita tapoja ratkaista vaikutuskaavioita, joissa on ongelman sisäisiä ja ulkopuolisia rajoitusehtoja sekä useita päätösvaiheita. Ratkaisuja ei kuitenkaan juurikaan löydy vaikutuskaavioille, joissa edellisiä päätöksiä ei muisteta, tai jotka sisältävät kokonaisvaltaisia rajoitteita. Decision Programming on yleinen optimointiviitekehys, joka on kykenevä ratkaisemaan edellä mainitut ongelmavariantit. Nämä päätösongelmat voidaan esittää Decision Programmingin avulla MILP-muodossa, joiden ratkaisuun löytyy monia algoritmeja. Tämä viitekehys ei kuitenkaan skaalaudu hyvin, joten sillä ei voida ratkaista kovin isoja ongelmia. Tämän kandidaatintyön tavoitteena oli kehittää approksimatiivinen algoritmi ongelman koon pienentämiseksi, jotta Decision Programming voi ratkaista ongelman nopeammin. Testasimme algoritmin toimintaa sensori-turbiini-systeemin huoltostrategian optimointiin. Algoritmi toimi tässä raportoidussa esimerkissä kohtuullisen hyvin, ja pienensi ongelman kokoa 95% menettäen alle 0.5% odotusarvollisesta hyödystä.

Avainsanat Vaikutuskaavio, Decision Programming, stokastinen optimointi, approksimatiivinen algoritmi

Contents

1 Introduction

Influence diagrams are extensively used in decision analysis to represent multi-stage decision problems under exogenous and endogenous uncertainty (Diehl and Haimes, 2004; Howard and Matheson, 2005). These diagrams are directed, acyclic graphs, whose nodes correspond to chance events, decisions, and values. Usually, the goal of an optimization problem illustrated by an influence diagram is to find a decision strategy that maximizes the expected value of a utility function over the different realizations of value nodes. Several approaches for solving influence diagrams have been presented. The most common include node removals and arc reversals (Shachter, 1986, 1988), and dynamic programming approaches (Tatman and Shachter, 1990).

However, the literature on Limited Memory Influence Diagrams (LIMIDs), in which the no-forgetting assumption does not hold, is sparse. Furthermore, the well-established techniques for solving influence diagrams are not able to include some common risk measures, such as Value at Risk (VaR). To address these challenges, Salo et al. (2019) develop the Decision Programming framework. This framework uses the structure of influence diagrams to represent the optimization problem mathematically and subsequently converts it into a corresponding Mixed Integer Linear Programming (MILP) problem. The framework is general and helps solve LIMIDs with several decision stages, multiple objectives, and endogenous uncertainties.

Decision Programming, however, has its drawbacks. As is typical for MILP-problems, the computational complexity increases roughly exponentially with problem size. Thus, the existing framework cannot readily handle problems with more than a few dozen decision and chance nodes. As real-life applications may have hundreds of nodes, this framework is not well-equipped to solve problems of considerable size in an acceptable amount of time.

In this thesis, we develop an approximation algorithm for Decision Programming. The algorithm will be used as a pre-solver, decreasing the problem size using probability and utility considerations. The goal of the research was to increase the applicability of the Decision Programming framework to solve problems of a larger scale with reasonable accuracy.

This thesis is structured as follows. Section 2 explores earlier approaches. Section 3 gives an introduction to influence diagrams and the Decision Programming framework. Section 4 describes the principles of the algorithm. Section 5 studies computational performance with an illustrative example. Section 6 concludes the thesis, with Appendix A presenting some proofs.

2 Earlier Approaches

Influence diagrams have been one of the most popular ways of visualizing multi-stage decision problems for nearly 50 years (see, e.g., Olmsted, 1984; Howard et al., 2006). As a result, the influence diagram representation has been extensively studied, and several ways of solving standard influence diagrams have been proposed. The most widely used of these are using node removals and arc reversals to iteratively simplify

the diagram (Olmsted, 1984; Shachter, 1986), and representing the diagram as a decision tree, solvable by dynamic programming, as described by Tatman and Shachter (1990).

Nevertheless, there are several situations in which information about earlier decisions is not available when making later ones. These kinds of influence diagrams are often called Limited Memory Influence Diagrams (LIMIDs), discussed by, e.g., Zhang et al. (1994). For LIMIDs, the well-established approaches of node removals, arc reversals, and dynamic programming are not applicable. Lauritzen and Nilsson (2001) present an iterative scheme for updating policies by deriving a junction tree from the limited memory influence diagram. Hovgaard and Brincker (2016) give an application example of LIMIDs for protecting against structural damage. Further approaches are described by Mauá et al. (2012). These approaches, however, are not equipped for solving problems with constraints spanning across the entire problem, for example, Conditional Value at Risk (CVaR).

One widely used method for incorporating several decision stages to optimization problems is stochastic programming, as in Zhou et al. (2013). Nevertheless, the realm of endogenous uncertainty in these problems has been largely unexplored. Some approaches are proposed by Rubinstein and Shapiro (1993). However, the above methods are not able to solve multi-stage limited memory problems with multiple objectives, problem-spanning constraints, and endogenous uncertainty.

The Decision Programming framework developed by Salo et al. (2019) is an exact solution method for solving LIMIDs, which is general enough to incorporate all the problem variants discussed above. The only restriction is that each chance and decision must have a finite set of discrete realizations. Incorporation of continuous random variables is discussed by Herrala (2020).

The main challenge with decision programming is, however, scalability. As discussed by Salo et al. (2019), larger real-life problems soon become intractable with Decision Programming. This thesis addresses this challenge by introducing an algorithm for pruning paths of a small probability from the problem, significantly reducing the problem size. We aim to improve the computational time of solving problems with Decision Programming so that it can be used to provide accurate results for larger real-life problems.

3 Introduction to Decision Programming

This section summarizes the Decision Programming framework. We closely follow the notation in Salo et al. (2019) but use a computationally faster problem statement.

An influence diagram is a directed, acyclic network $G = (N, A)$. It has three types of nodes $N = C \cup D \cup V$: chance nodes C, decision nodes D, and value nodes V. Chance nodes C correspond to random variables, representing uncertain events, decision nodes D represent decisions, and value nodes V express consequences resulting from the realizations of both the random variables of C and decisions made at D.

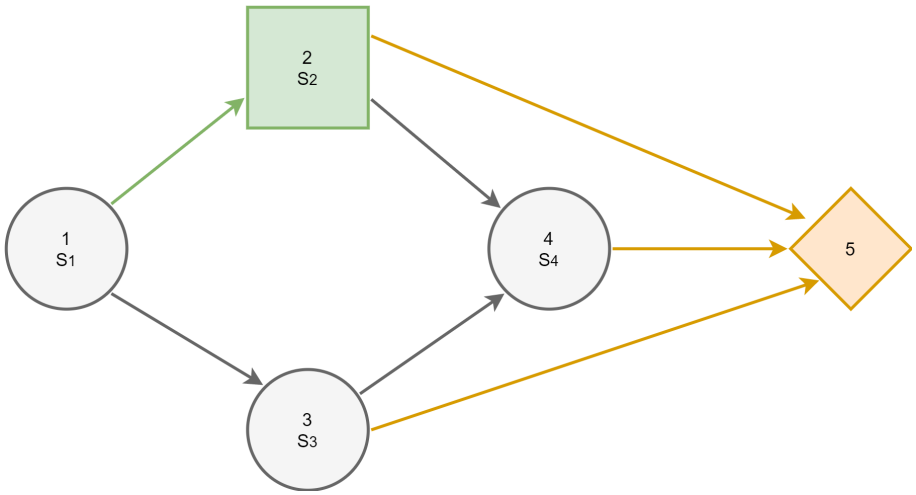

Figure 1: An example of an influence diagram

Figure 1 shows an example of an influence diagram. Here the circles represent chance nodes C, squares decision nodes D, and diamonds value nodes V. The numbers show the indexing of the nodes. This influence diagram could, for example, depict the decision problem of choosing whether to take an umbrella. Here node 1 would be the weather forecast, 2 the decision to take the umbrella, and 3 the actual weather. Node 4 could describe the decision maker's wetness level, while the utility node 5 would be their mood.

Arcs $A = \{(i,j) \,|\, i,j \in N\}$ represent dependencies between nodes. For $j \in C, D, V$, these arcs represent how earlier realizations influence probability distributions, decisions to be made, and values, respectively. Let us define the information set of a node $j \in N$ as $I(j) = \{i \in N \,|\, (i,j) \in A\}$; thus, the nodes i, from which there is an arc to j. As the network is acyclic, we can index the chance and decision nodes as $C \cup D = \{1, 2, ..., n\}$, where $\forall\,(i,j) \in A : i < j$. Furthermore, let us index the value nodes as $V = \{n+1, n+2, ..., n+|V|\}$.

All nodes $j \in C \cup D$ have a finite number of discrete states $s_j \in S_j$. For a chance node, these states represent the possible realizations of random variables, and for a decision node, the decision alternatives. Let us denote by X_j the random and decision variables governing the realization of states. A path $s = (s_1, s_2, ..., s_n)$ is defined as a sequence of states $s_j \in S_j$. Thus, a path is a combination of states of all the chance and decision nodes. A given state k of node j is denoted by s_{jk}. Furthermore, for any subset of nodes $J \subseteq C \cup D$, we denote by $s_J \in S_J$ a subpath, for which the states of nodes J are set.

Let also $Z = (Z_1, ..., Z_n) \in \mathbb{Z}$, where $Z_j : s_{I(j)} \mapsto s_j$, be a decision strategy; each Z_j maps every possible information state $I(j)$ of node $j \in D$ to a decision, i.e., a state. Here $\mathbb{Z}$ is the set of all decision strategies. A decision strategy Z is compatible with a path s, if it maps every possible information state of nodes $j \in D$ to a decision, which is also included in s. Furthermore, let $z(s_j \,|\, s_{I(j)}) \in \{0, 1\}$ be a binary variable, whose value shows whether the decision strategy is (1), or is not (0), compatible at node j. In other words, $Z_j(s_{I(j)}) = s_j \iff z(s_j \,|\, s_{I(j)}) = 1$.

Assuming a compatible decision strategy, the probability of a subpath s_k, which consists of nodes $1, ..., k$, can be expressed as

$$p(s_k) = \prod_{\substack{i \in C \\ i \leq k}} \mathbb{P}(X_i = s_i \,|\, X_{I(j)} = s_{I(j)}), \tag{1}$$

with $p(s)$ being the probability of a full path. Let us, for simplicity, assume that there is a single value node $v \in V$. Thus, the utility gained with path s can be denoted by $\mathcal{U}(s)$, assuming that a well-defined function maps the information set of the value node to a set of consequences. Here, the utility function is constructed to map to the unit interval $[0, 1]$.

What follows is a computationally improved version of Decision Programming. We shall define $\overline{C}$ and $\overline{D}$ as

$$\overline{C} = C \cup \{k \in C \cup D \,|\, \exists\, j \in C : k \in I(j)\}$$
$$\overline{D} = D \cup \{k \in C \cup D \,|\, \exists\, j \in D : k \in I(j)\}.$$

Thus, $\overline{C}$ is the union of all chance nodes and their information sets; $\overline{D}$ is a similar union for decision nodes. Furthermore, let $S_{\overline{C}}$ consist of all subpaths $s_{\overline{C}} = (s_{i_1}, ..., s_{i_{\overline{C}}})$, such that $i_k \in \overline{C}\ \forall\ k = 1, ..., |\overline{C}|$; thus, the set of subpaths consisting only of the nodes in $\overline{C}$. Sets of subpaths $S_{\overline{D}}$ and $S_{\overline{V}}$ are defined similarly. Let $x(s_{\overline{D}}) \in \{0, 1\}$ be a binary decision variable, whose value is 1, if and only if the decision strategy is compatible at every node $j \in D$; 0 otherwise. Using equation 1, we define $p(s_{\overline{C}}) = \prod_{i \in C} p(s_i | s_{I(i)}) = p(s)$. Now, the improved Decision Programming MILP-formulation may be written as:

$$\max_{Z \in \mathbb{Z}} \sum_{s \in S} x(s_{\overline{D}}) p(s_{\overline{C}}) \mathcal{U}(s_{\overline{V}}) \tag{2}$$

$$\text{subject to} \sum_{s_j \in S_j} z(s_j \,|\, s_{I(j)}) = 1 \quad \forall j \in D, s_{I(j)} \in S_{I(j)} \tag{3}$$

$$1 - \Big[|D| - \sum_{j \in D} z(s_j | s_{I(j)})\Big] \leq x(s_{\overline{D}}) \leq \frac{1}{|D|} \sum_{j \in D} z(s_j | s_{I(j)}), \quad \forall s_{\overline{D}} \in S_{\overline{D}} \tag{4}$$

$$z(s_j \,|\, s_{I(j)}) \in \{0, 1\}, \quad \forall j \in D, s_j \in S_j, s_{I(j)} \in S_{I(j)} \tag{5}$$

$$x(s_{\overline{D}}) \in \{0, 1\}, \quad \forall s_{\overline{D}} \in S_{\overline{D}}. \tag{6}$$

Here, (2) is the expected value of utility to be maximized, constraints (3) and (4) enforce the decision strategies to be compatible, while (5) and (6) give the range for variables x and z.

4 Methodological Development

This section develops the mathematical foundations of our algorithm. This development assumes that the influence diagram has a single value node. However, this

algorithm can easily be adapted to problems concerning multiple value nodes. We begin by making some definitions concerning utility:

$$\overline{\mathcal{U}}(s_J) = \max_{s' \in S_{I(v)\setminus J}} \mathcal{U}(s' \cup s_{J \cap I(v)}) \tag{7}$$

Thus, $\overline{\mathcal{U}}(s_J)$ is the maximum amount of utility that can be achieved when the states of nodes J are set to s_J. Let also $p(s_J)$ be the path probability of the subpath $s_J = (s_1, ..., s_{|J|})$, calculated as in equation (1); here $J = \{1, ..., k\}$. Furthermore, let us define the upper bound for the effect of paths starting with s_J as

$$\overline{\mathbb{E}}_{\mathcal{U}}(s_J) = p(s_J)\overline{\mathcal{U}}(s_J).$$

Let $S_J = \{s_{J_1}, s_{J_2}, ..., s_{J_m} \,|\, \nexists\, s : s_{J_k}, s_{J_l} \subseteq s\}$ be a mutually exclusive set of subpaths. Thus, the nodes which are set might be the same, as long as the states differ. Furthermore, let $\overline{S}_J = \left\{s' \in S \,|\, \exists\, k : s'_{J_k} = s_{J_k}\right\}$ be the set of full path extensions of the subpaths in S_J. Now we can make the following definition:

$$\mathbb{E}^*_{\mathcal{U}}(S_J) = \max_{Z \in \mathbb{Z}} \sum_{s \in S \setminus \overline{S}_J} x(s_{\overline{D}}) p(s_{\overline{C}}) \mathcal{U}(s_{\overline{V}})$$

Thus, $\mathbb{E}^*_{\mathcal{U}}(S_J)$ is the optimal solution to the optimization problem considering only the paths that are not extensions of $s_{J_k} \in S_J$.Furthermore, we denote the optimal solution to the original optimization problem as $\mathbb{E}^*_{\mathcal{U}}$.

Now, the applicability of our algorithm builds on the following theorem:

Theorem 1. *Solving the optimization problem with only the paths not starting with $s_{J_k} \in S_J$ gives the following interval approximation for the optimal solution of the original problem:*

$$\mathbb{E}^*_{\mathcal{U}} = \left[\mathbb{E}^*_{\mathcal{U}}(S_J)\,,\, \mathbb{E}^*_{\mathcal{U}}(S_J) + \sum_{k=1}^{m} \overline{\mathbb{E}}_{\mathcal{U}}(s_{J_k})\right].$$

Thus, the true optimum will be inside this interval.

Proof. See Appendix A.

Let us still make a few more definitions. First, we shall define S_ε as follows:

$$S_\varepsilon = \{s \in S \,|\, p(s) > \varepsilon\}.$$

Thus, a path belongs in S_ε if and only if its path probability is greater than ε. We will hereafter call S_ε as the set of significant paths, and correspondingly a path s as significant, if $s \in S_\varepsilon$.

Furthermore, let $\overline{\delta}$ be the upper bound on the effect on expected utility of taking only paths in S_ε into account. Using the notation from before, $S_\varepsilon = S \setminus \overline{S}_J$. Thus, using Theorem 1, this effect can be written as

$$\overline{\delta} = \sum_{s \notin S_\varepsilon} \overline{\mathbb{E}}_{\mathcal{U}}(s). \tag{8}$$

Finally, let n_j be the number of states for node j. With these definitions, we can begin the development of our algorithm.

We first form a tree structure of all the possible paths of the problem. As subpath probability cannot increase with path length, we can employ a depth-first search algorithm on the obtained decision tree. If, at any point, $p(s_J) \leq \varepsilon$, we abandon all the paths starting with s_J, as their total probability is at most ε. Here we will update $\overline{\delta}$ iteratively, for every subpath s_J we abandon, according to (8). If we arrive at the last node $j = n$ with a path such that $p(s) > \varepsilon$, we will add it to the set of significant paths S_ε. When the algorithm has explored the whole tree, it will return the set of significant paths, S_ε, and the upper bound for the loss of expected utility, $\overline{\delta}$.

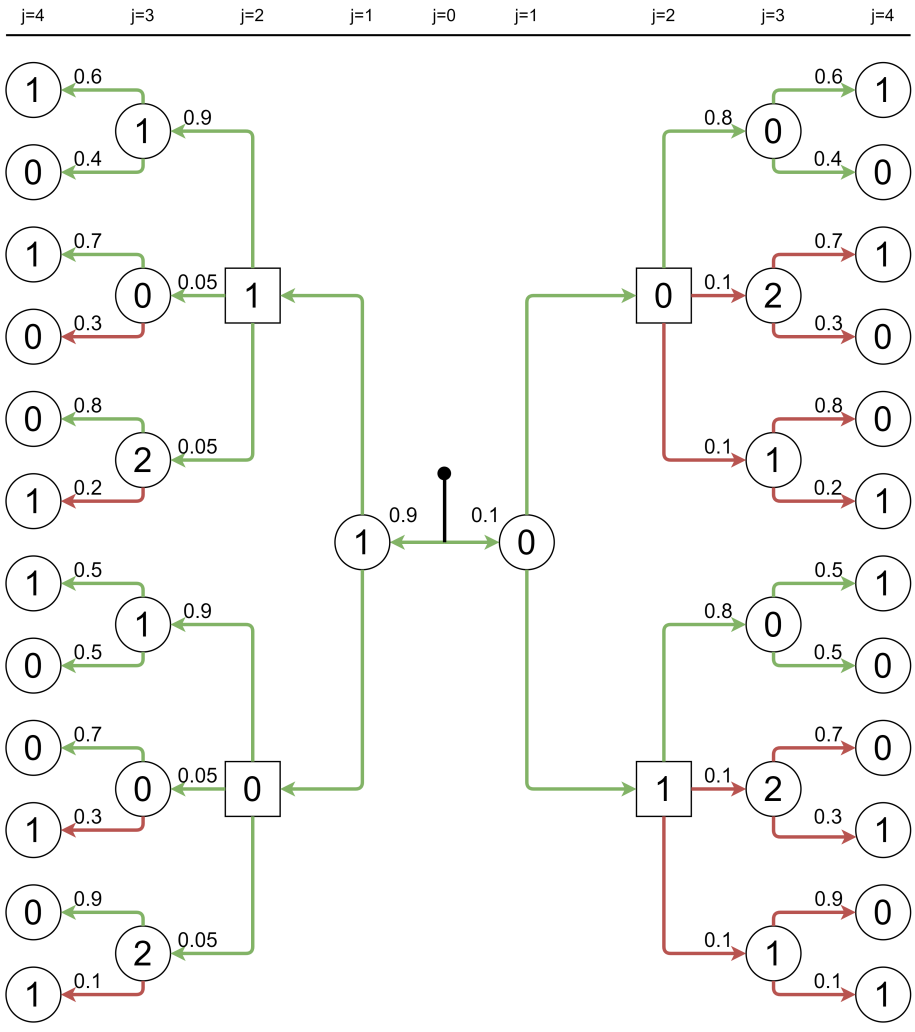

Figure 2: Tree representation of a decision problem

Figure 2 shows a tree representation of the problem depicted by the influence diagram in Figure 1. In this problem, the states for each node are:

$$S_1 = \{0, 1\}$$
$$S_2 = \{0, 1\}$$
$$S_3 = \{0, 1, 2\}$$
$$S_4 = \{0, 1\},$$

with $\varepsilon = 0.02$. As described previously, the algorithm performs a depth-first search, while pruning the rest of subpaths, whose probability does not exceed ε. The pruned subpaths are depicted by the red arrows.

The approximation algorithm is now given by the following pseudocode.

procedure PRE-SOLVER ▷
  $j \leftarrow 0$ ▷ Index of the current node, $j \in C \cup D$
  $g[l] \leftarrow n_l, l = 1, ..., n$ ▷ Vector of amount of states for each node
  $k[l] \leftarrow 1, l = 1, ..., n + 1$ ▷ $k(l)$ shows the state of node l we are currently on
  $S_\varepsilon \leftarrow \emptyset$ ▷ The set of significant paths
  $s_J \leftarrow \emptyset$ ▷ The set of node states which are set
  $\overline{\delta} \leftarrow 0$ ▷ The amount of expected utility lost at most
  $q \leftarrow 1$ ▷ Probability of current path
  while $j > 0 \vee k[1] \leq g[1]$ **do**
    if $j = n \vee k[j + 1] > g[j + 1] \vee q \leq \varepsilon$ **then**
      if $q \leq \varepsilon$ **then** ▷ Probability of path low
        $\overline{\delta} \leftarrow \overline{\delta} + q\overline{\mathcal{U}}(s_J)$ ▷ Increase the max. missed utility counter
      else
        if $j = n$ **then** ▷ At last node
          $S_\varepsilon \leftarrow S_\varepsilon \cup s_J$ ▷ Append path to set of paths
      $s_J \leftarrow s_J \setminus s_{jk[j]}$ ▷ Substract node from path
      if $j \in C$ **then** ▷ Current node is a chance node
        $q \leftarrow q / p(s_{jk[j]} \mid S_{I(j)})$ ▷ Update probability
      $k[j] \leftarrow k[j] + 1$ ▷ Increase state counter
      $j \leftarrow j - 1$ ▷ Move up a node
      $k[l] \leftarrow 1, l = j + 2, ..., n + 1$ ▷ Initialize nodes down the tree again
    else
      $j \leftarrow j + 1$ ▷ Move down a node
      $s_J \leftarrow s_J \cup s_{jk[j]}$ ▷ Append current node state to path
      if $j \in C$ **then** ▷ Current node is a chance node
        $q \leftarrow q \cdot p(s_{jk[j]} \mid S_{I(j)})$ ▷ Update probability
  return $(S_\varepsilon, \overline{\delta})$

5 Results

The computational example of this thesis comes from a prognostics problem in Mancuso et al. (2021). We first overview the model described in the paper and then

present the results of pruning paths with our algorithm. Finally, we discuss the relevance of the algorithm.

5.1 Problem Description

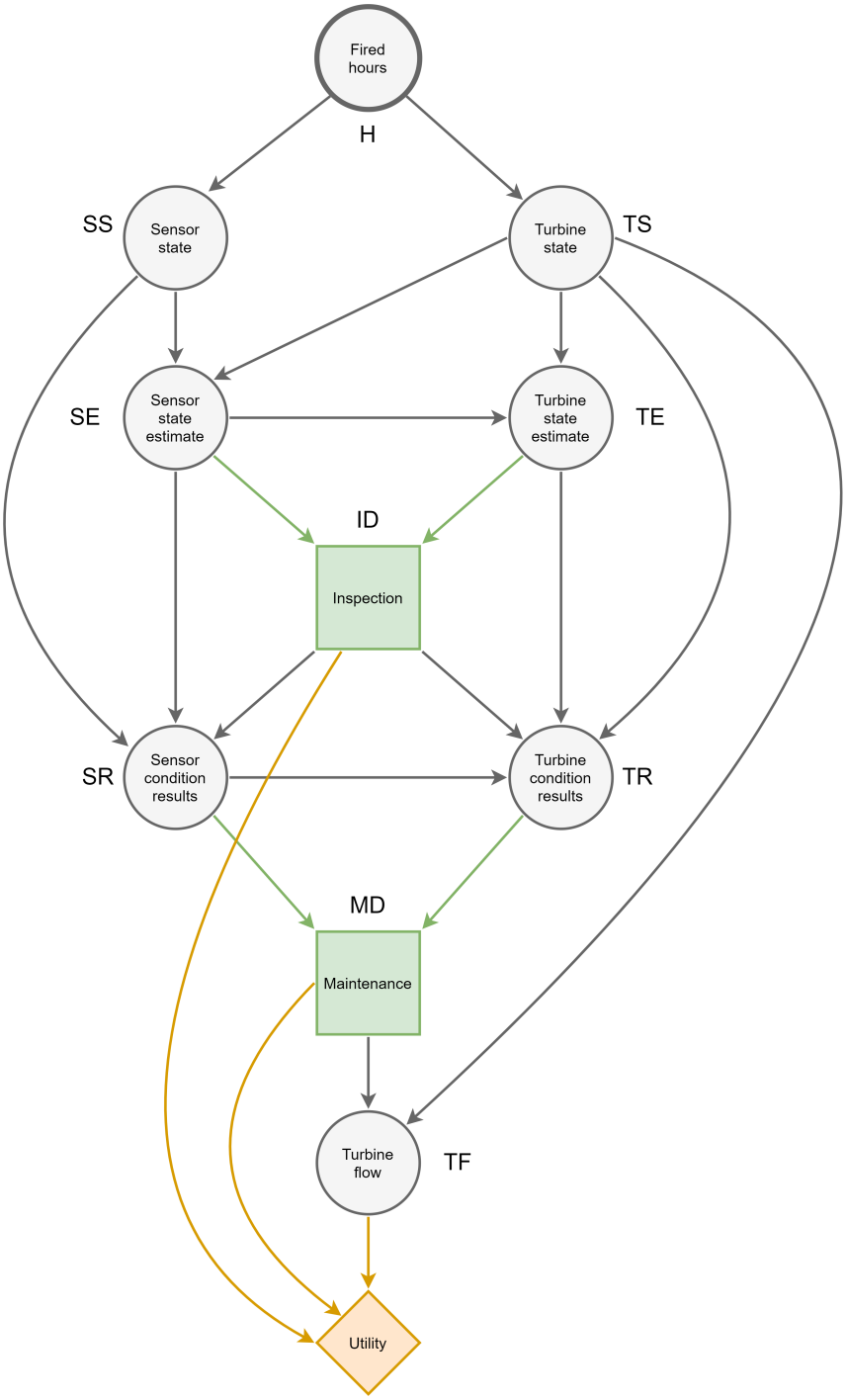

Figure 3: Influence diagram of turbine inspection and maintenance

The case study in question consists of a gas turbine equipped with a Prognostics and Health Management (PMH) solution. The turbine has several sensors monitoring the state of various components of the turbine. In addition, the system has sensor validation algorithms monitoring the state of both the sensors and the turbine. We assume that global data from the validation algorithms are available; thus, we have single condition estimates for the turbine and the sensors.

An influence diagram of the system described above is shown in Figure 3. Here, the letters outside the nodes denote the indexing of the nodes. As previously, circles represent chance nodes, squares decision nodes, and the diamond is the value node.

The first node of the graph, $s_H \in \{0, 4000, 8000, 12000...\}$, is referred to as fired hours. The chance node s_H is deterministic, where probability is one for its current state. Both the states of the sensor and the turbine,

$$s_{SS}, s_{TS} \in \{Excellent, Good, Fair, Poor, Failing\},$$

depend on the value of s_H; the probability of the turbine and the sensor being in a worse state grows with fired hours. Now, the PHM solution uses machine learning to come up with predictions for the actual sensor and turbine states, s_{SE} and s_{TE}. The sensor state estimate depends on the turbine state as well as the actual sensor state. This estimate, in turn, influences the turbine state estimate combined with the actual state of the turbine. The estimates have the same discretization as the sensor and turbine states.

Now, these state estimates influence the inspection decision:

$$s_{ID} \in \{None, SensorCheck, ConditionMonitoring\}.$$

SensorCheck implies the detection of faulty sensors and ConditionMonitoring their repair. The inspection decision and previous estimates lead to condition results

$$s_{SR}, s_{TR} \in \{Excellent, Good, Fair, Poor, Failing\}.$$

If $s_{ID} = None$, $s_{SR} = s_{SE}$ and $s_{TR} = s_{TE}$. If the inspection decision is a sensor check, the sensor condition results are correct with a high probability, and the turbine condition results depend on the sensor condition results. Finally, if the inspection is condition monitoring, both the sensor and turbine condition are correct with a high probability, governed by the problem-specific parameters.

Now, based on s_{TR} and s_{SR} we have a Maintenance Decision to make:

$$s_{MD} = \{None, Level1, Level2\}.$$

A Level1 maintenance decision returns the state of the turbine to 4000 fired hours earlier, while a Level2 maintenance restores the system to practically 0 hours of operation. Thus, the node s_H is not the time the system has been operating, but a combination of operation time and the maintenance decisions made.

Finally, the maintenance decision made with the actual turbine state effect turbine flow s_{TF}, which has the same discretization as the other state nodes. The final utility depends on the quality of turbine flow together with the costs of the inspection and maintenance measures. The goal of this model is to maximize the expected utility of the system.

5.2 Computations

For one set value of fired hours, the system depicted in the previous subsection has a total of $|S| = 5^7 \cdot 3^2 = 703125$ paths. The Decision Programming Framework took roughly 26 hours to solve this problem. We began the analysis by setting $\varepsilon = 0$; thus, pruning only the paths with probability zero and arriving at an exact answer. This process yielded $|S_\varepsilon| = 78415$. Thus, the problem size was pruned by nearly 89%. This computation took our algorithm roughly 30 seconds. As solving a MILP-problem is NP-hard, the widely accepted assumption $P \neq NP$ means that the problem generally scales worse than linearly. This means that this reduction in problem size leads to more than a 90% reduction in computation time. This reduction is 3 orders of magnitude larger than the time taken by our algorithm ($\frac{0.9 \cdot 26h}{30s} \approx 3 \cdot 10^3$). Furthermore, our algorithm also scales linearly with the number of paths, which means that this effect is amplified with a larger problem size.

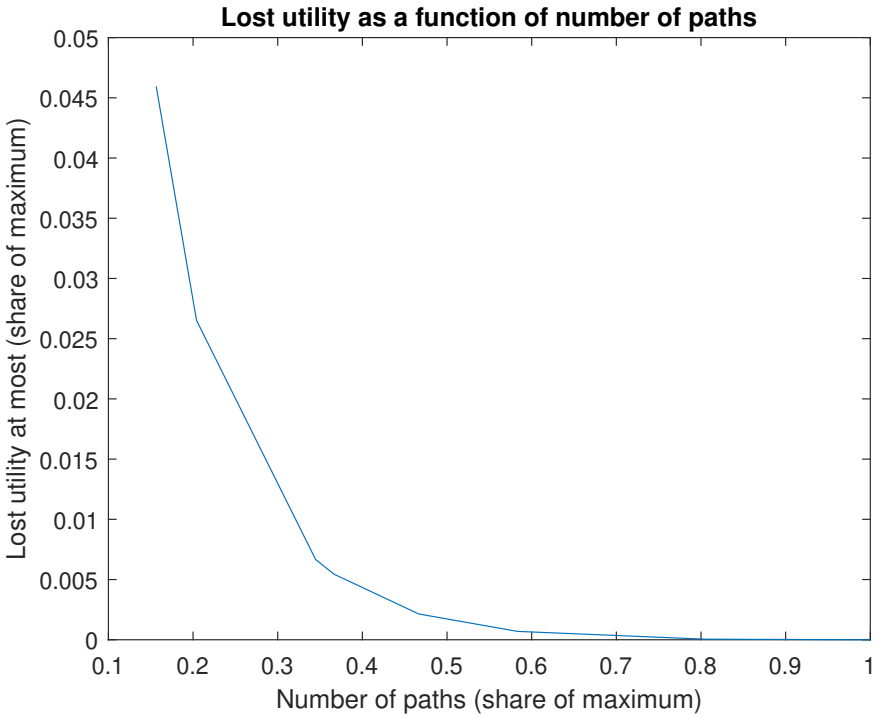

Figure 4: $\overline{\mathcal{U}}$ as a function of $|S_\varepsilon|$

The graph in Figure 4 shows results of further pruning. Here we pruned the problem by setting different values of ε and plotting the subsequent maximum utility loss $\overline{\mathcal{U}}$ as a function of the number of paths. Here the maximum utility lost at most is the optimum found by Decision Programming, while the maximum number of paths is after pruning 0-probability paths, i.e., 78415.

We can see from the graph that by further pruning 20% of the remaining paths, the maximum lost utility is less than a tenth of a percent of the optimum. This percentage would be considered an acceptable amount lost, as the model's assumptions have some inaccuracies.

Furthermore, by pruning 60% of these paths, we lose at most 0.5% of expected utility. Pruning more than this is probably not worth it, as the lost utility grows roughly exponentially with the number of paths we prune.

In the context of this particular problem, the results are promising. By reducing the problem size by more than 95%, we arrive at an approximation at most 0.5% away from the optimum.

6 Conclusion

Decision Programming is an optimization framework capable of solving various kinds of problems. The framework, however, does not scale well with problem size. The goal of this research was to introduce an algorithm for generating the most probable paths in a decision problem so that Decision Programming can arrive at an approximate solution faster.

The performance of the algorithm was tested on a Prognostics and Health Management problem. It pruned 90% of the number of paths without any loss of accuracy and 96% with less than 0.5% loss in expected utility. Furthermore, the computation time of our algorithm was more than three orders of magnitude less than The Decision Programming framework took solving the original problem. As the solving process in this problem scales worse than linearly, this is a good result.

There are, however, several limitations in our study. First, this example is not indicative of performance in other problems. Our algorithm generally performs better as the variance of the probabilities of the problem grows. Thus, in other problems, our algorithm may not reduce the problem size, and its usage may thus not be worthwhile. Furthermore, the exact computational advantage of the algorithm, even in this particular problem, cannot be assessed without solving the pruned problem with Decision Programming.

Further research should focus primarily on the issues mentioned above: testing the algorithm's performance across a variety of problems and implementing it to be a part of the Decision Programming framework. Furthermore, the code can be optimized in several ways. First, the probability transitions can be ordered, allowing for faster pruning from the same node level. Second, the algorithm can be supplemented with further heuristics to prune out decision and chance nodes. Lastly, the algorithm and Decision Programming might be improved by dividing the problem into parts and using dynamic programming, when applicable, to speed up the computations.

References

Michael Diehl and Yacov Y Haimes. Influence diagrams with multiple objectives and tradeoff analysis. *IEEE Transactions on Systems, Man, and Cybernetics-Part A: Systems and Humans*, 34(3):293–304, 2004.

Olli Herrala. An efficient strategy for solving stochastic programming problems under endogenous and exogenous uncertainties. Master's thesis. 2020. URL http://urn.fi/URN:NBN:fi:aalto-202003222575.

Mads K Hovgaard and Rune Brincker. Limited memory influence diagrams for structural damage detection decision-making. *Journal of Civil Structural Health Monitoring*, 6(2):205–215, 2016.

Ronald A Howard and James E Matheson. Influence diagrams. *Decision Analysis*, 2 (3):127–143, 2005.

Ronald A Howard, James E Matheson, Miley W Merkhofer, Allen C Miller, and D Warner North. Comment on influence diagram retrospective. *Decision Analysis*, 3(2):117–119, 2006.

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

Alessandro Mancuso, Michele Compare, Ahti Salo, and Enrico Zio. Optimal prognostics and health management-driven inspection and maintenance strategies for industrial systems. *Reliability Engineering & System Safety*, 210:107536, 2021.

Denis Deratani Mauá, Cassio Polpo De Campos, and Marco Zaffalon. Solving limited memory influence diagrams. *Journal of Artificial Intelligence Research*, 44:97–140, 2012.

Scott Mostyn Olmsted. *On Representing and Solving Decision Problems*. PhD thesis, Stanford University, 1984.

Reuven Y Rubinstein and Alexander Shapiro. *Discrete event systems: Sensitivity analysis and stochastic optimization by the score function method*, volume 13. Wiley, 1993.

Ahti Salo, Juho Andelmin, and Fabricio Oliveira. Decision programming for multi-stage optimization under uncertainty. *arXiv preprint arXiv:1910.09196*, 2019.

Ross D Shachter. Evaluating influence diagrams. *Operations Research*, 34(6): 871–882, 1986.

Ross D Shachter. Probabilistic inference and influence diagrams. *Operations Research*, 36(4):589–604, 1988.

Joseph A Tatman and Ross D Shachter. Dynamic programming and influence diagrams. *IEEE transactions on Systems, Man, and Cybernetics*, 20(2):365–379, 1990.

Nevin Lianwen Zhang, Runping Qi, and David Poole. A computational theory of decision networks. *International Journal of Approximate Reasoning*, 11(2):83–158, 1994.

Yang Zhou, Guo H Huang, and Boting Yang. Water resources management under multi-parameter interactions: A factorial multi-stage stochastic programming approach. *Omega*, 41(3):559–573, 2013.

A Proofs

Proof of Theorem 1.

Let $s_J \in S_J$. We shall denote by $\overline{S}(s_J)$ the paths which are full path extensions of s_J. We can partition the sum in (2) based on whether the path is an extension of s_J:

$$\sum_{s \in S} x(s_{\overline{D}}) p(s_{\overline{C}}) \mathcal{U}(s_{\overline{V}}) = \sum_{s \in S \setminus \overline{S}(s_J)} x(s_{\overline{D}}) p(s_{\overline{C}}) \mathcal{U}(s_{\overline{V}}) + \sum_{s \in \overline{S}(s_J)} x(s_{\overline{D}}) p(s_{\overline{C}}) \mathcal{U}(s_{\overline{V}}). \quad \text{(A1)}$$

We shall seek an upper bound for the second term in (A1).

First, (6) gives $x(s_{\overline{D}}) \leq 1$. As all paths in the second term are full path extensions of s_J, definition (7) implies $\mathcal{U}(s_{\overline{V}}) \leq \overline{\mathcal{U}}(s_J)$. Finally, we can write the probability of any path as

$$\prod_{i \in C} p(s_i | s_{I(i)}) = \prod_{\substack{i \in C \\ i \leq j}} p(s_i | s_{I(i)}) \prod_{\substack{i \in C \\ i > j}} p(s_i | s_{I(i)}).$$

As the states for nodes J are fixed, the first product is just equal to $p(s_J)$. With the preceding remarks, we arrive at the following upper bound:

$$\begin{aligned} \sum_{s \in \overline{S}(s_J)} x(s_{\overline{D}}) p(s_{\overline{C}}) \mathcal{U}(s_{\overline{V}}) &\leq \sum_{s \in \overline{S}(s_J)} p(s_{\overline{C}}) \mathcal{U}(s_{\overline{V}}) \leq \overline{\mathcal{U}}(s_J) \sum_{s \in \overline{S}(s_J)} p(s_{\overline{C}}) \\ &= \overline{\mathcal{U}}(s_J) p(s_J) \sum_{s \in \overline{S}(s_J)} \prod_{\substack{i \in C \\ i > j}} p(s_i | s_{I(i)}) = p(s_J) \overline{\mathcal{U}}(s_J), \end{aligned}$$

where the last equality follows from the fact that the paths are mutually exclusive and commonly exhaustive, i.e., the sum of their probabilities is 1.

Furthermore, as the summands are all contributions to the expected value of the utility, which is scaled to the unit interval, they are all non-negative. Thus, a lower bound for the second term is just 0.

As all full paths are mutually exclusive, the following contributions for each $s_J \in S_J$ can be summed together. Here we take the first term in (A1) to exclude full paths starting with all $s_J \in S_J$. Thus, we arrive at the following interval approximation for the expected utility:

$$\mathbb{E}^*_{\mathcal{U}} = \left[\mathbb{E}^*_{\mathcal{U}}(S_J) \,,\, \mathbb{E}^*_{\mathcal{U}}(S_J) + \sum_{k=1}^{m} \overline{\mathbb{E}}_{\mathcal{U}}(s_{J_k}) \right]. \square$$