A Framework for Integrating Influence Diagrams and POMDPs

Jinchuan Shi

Follow this and additional works at: https://scholarsjunction.msstate.edu/td

Recommended Citation
https://scholarsjunction.msstate.edu/td/134

This Dissertation is brought to you for free and open access by the Theses and Dissertations at Scholars Junction. It has been accepted for inclusion in Theses and Dissertations by an authorized administrator of Scholars Junction. For more information, please contact scholcomm@msstate.libanswers.com.
A framework for integrating influence diagrams and POMDPs

By

Jinchuan Shi

A Dissertation
Submitted to the Faculty of
Mississippi State University
in Partial Fulfillment of the Requirements
for the Degree of Doctor of Philosophy
in Computer Science
in the Department of Computer Science and Engineering

Mississippi State, Mississippi

May 2018
Copyright by

Jinchuan Shi

2018
A framework for integrating influence diagrams and POMDPs

By

Jinchuan Shi

Approved:

Eric Hansen
(Major Professor)

Ioana Banicescu
(Committee Member)

J. Edward Swan II
(Committee Member)

Maxwell Young
(Committee Member)

T. J. Jankun-Kelly
(Graduate Coordinator)

Jason M. Keith
Dean
Bagley College of Engineering
An influence diagram is a widely-used graphical model for representing and solving problems of sequential decision making under imperfect information. A closely-related model for the same class of problems is a partially observable Markov decision process (POMDP). This dissertation leverages the relationship between these two models to develop improved algorithms for solving influence diagrams.

The primary contribution is to generalize two classic dynamic programming algorithms for solving influence diagrams, Arc Reversal and Variable Elimination, by integrating them with a dynamic programming technique originally developed for solving POMDPs. This generalization relaxes constraints on the ordering of the steps of these algorithms in a way that dramatically improves scalability, especially in solving complex, multi-stage decision problems.

A secondary contribution is the adoption of a more compact and intuitive representation of the solution of an influence diagram, called a strategy. Instead of representing a
strategy as a table or as a tree, a strategy is represented as an acyclic graph, which can be
exponentially more compact, making the strategy easier to interpret and understand.

Key words: Influence Diagram, POMDP, Graphical Model, Theoretical Decision Planning,
Probabilistic Inference.
DEDICATION

To my wife, Xi Chen, my daughter Katherine C. Shi, and my parents, Xinya Shi and Yaqin Song.
ACKNOWLEDGEMENTS

First and foremost, I am indebted to my advisor and mentor, Dr. Eric Hansen, who has played a key role by his continuous support, his invaluable guidance and his insightful advice for my research as well as many other aspects of my life. He helped me so much in many different ways including giving me the opportunity to pursue my Ph.D., pointing directions for my research, spending so much time discussing research with me, carefully reading and commenting an early draft of this dissertation and so on.

I would like to thank Dr. Banicescu, Dr. Swan, and Dr. Young for serving as my Ph.D. committee members and their precious suggestions of my research.

Finally, my special thanks go to my wife, Xi Chen, for her patience and continuous support. I would like to thank my parents, Xinya Shi and Yaqin Song, for their encouragement and moral support. I also would like to thank my daughter Katherine C. Shi, for her moral support.
# TABLE OF CONTENTS

**DEDICATION** ................................................................. ii

**ACKNOWLEDGEMENTS** ..................................................... iii

**LIST OF TABLES** ........................................................... vi

**LIST OF FIGURES** .......................................................... vii

**CHAPTER**

1. **INTRODUCTION** .......................................................... 1
   1.1 Motivation .......................................................... 1
   1.2 Contributions ..................................................... 2
   1.3 Dissertation Outline .............................................. 6

2. **BACKGROUND** ........................................................... 7
   2.1 Influence Diagrams ................................................ 7
      2.1.1 Model Descriptions and Notations ......................... 7
      2.1.2 Used Car Buyer Example ................................... 10
      2.1.3 Variables Independence Analysis ........................... 14
   2.2 Evaluating Influence Diagrams .................................... 18
      2.2.1 Strategy and Strategy Representation ....................... 18
      2.2.2 Probability Computations in Influence Diagrams ........ 19
      2.2.3 Maximum Expected Utility ................................. 22
      2.2.4 Classic Algorithms ......................................... 26
   2.3 Partially Observable Markov Decision Processes .............. 27
      2.3.1 Model Descriptions and Notations ......................... 28
      2.3.2 POMDP Strategy Representation ............................ 29
      2.3.3 Value Function Representation ............................. 30
      2.3.4 Eliminating Dominated Vectors ............................. 32
      2.3.5 Evaluating POMDPs ........................................... 34
      2.3.6 Incremental Pruning Algorithm ............................. 35
# LIST OF TABLES

| 5.1 | Comparison of the number of nodes in the strategy trees, and equivalent strategy graph, for the *CHILD* influence diagram. | 130 |
| 5.2 | Comparison of number of nodes in the strategy trees, and equivalent strategy graph, for the Arthonet influence diagram. | 135 |
LIST OF FIGURES

2.1 Partial ordering in an influence diagram ...................................... 9
2.2 Influence Diagram for Used Car Buyer Problem .............................. 10
2.3 Conditional Probability Tables (CPTs) for each of the chance nodes in Used Car Buyer Problem .......................................................... 11
2.4 The Utility Table for Used Car Buyer Problem ................................. 12
2.5 Serial Connections ........................................................................ 15
2.6 Diverging Connections ................................................................... 15
2.7 Converging Connections .................................................................. 15
2.8 The Example Influence Diagram EID ................................................ 21
2.9 The decision tree of ID EID ............................................................... 23
2.10 The decision tree of ID EID with $D_2$ replaced by $MEU_{D_2}$ ............... 24
2.11 Three-stage POMDP as influence diagram ........................................ 28
2.12 POMDPs value function representation ............................................ 31
3.1 Undominated strategies for the used car buyer ID, before selection of the optimal strategy for the initial belief ................................................. 69
3.2 Four Different Mazes ..................................................................... 81
3.3 Influence diagram of the maze navigation problem (graphical part) .... 81
3.4 Maze Strategy Graph ...................................................................... 83
3.5 Mildew problem influence diagram (a) and strategy graph (b) ............. 84
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.6</td>
<td>The Influence Diagram of the ALARM Monitoring System Problem</td>
<td>86</td>
</tr>
<tr>
<td>4.1</td>
<td>Barren Node Removal</td>
<td>89</td>
</tr>
<tr>
<td>4.2</td>
<td>Value Node Removal</td>
<td>89</td>
</tr>
<tr>
<td>4.3</td>
<td>Chance Node Removal</td>
<td>90</td>
</tr>
<tr>
<td>4.4</td>
<td>Decision Node Removal</td>
<td>91</td>
</tr>
<tr>
<td>4.5</td>
<td>Reverse the arc between two chance nodes</td>
<td>92</td>
</tr>
<tr>
<td>4.6</td>
<td>Example of ID before (left) and after (right) a decision node reduction.</td>
<td>97</td>
</tr>
<tr>
<td>4.7</td>
<td>Examples of observation node reduction</td>
<td>100</td>
</tr>
<tr>
<td>4.8</td>
<td>Influence diagram for oil wildcatter problem</td>
<td>105</td>
</tr>
<tr>
<td>4.9</td>
<td>Arc-Reversal algorithm performs on oil wildcatter problem with elimination ordering: Oil, Drill, Seismic, Test. (A) Initial the strategy node Strategy using last decision node Drill. (B) Apply Arc-Reversal between Oil and Seismic. (C) Remove Oil. (D) Remove Seismic. (E) Remove Test.</td>
<td>106</td>
</tr>
<tr>
<td>4.10</td>
<td>Arc-Reversal algorithm performs on oil wildcatter problem with elimination ordering: Drill, Seismic, Test, Oil. (A) Initial the strategy node Strategy using last decision node Drill. (B) Remove Seismic. (C) Remove Test. (D) Remove Oil.</td>
<td>107</td>
</tr>
<tr>
<td>5.1</td>
<td>Strategy tree for oil wildcatter problem</td>
<td>110</td>
</tr>
<tr>
<td>5.2</td>
<td>A Demonstrate Influence Diagram Example</td>
<td>113</td>
</tr>
<tr>
<td>5.3</td>
<td>The strategy graph created after eliminated variable $D_2$.</td>
<td>115</td>
</tr>
<tr>
<td>5.4</td>
<td>The strategy graph created after eliminated variable $Z_2$.</td>
<td>116</td>
</tr>
<tr>
<td>5.5</td>
<td>The strategy graph created after eliminated variable $D_1$.</td>
<td>117</td>
</tr>
<tr>
<td>5.6</td>
<td>The strategy graph created after eliminated variable $Z_1$.</td>
<td>118</td>
</tr>
</tbody>
</table>
5.7 Merge branches with the same predecessor and successor nodes ............... 120
5.8 Remove irrelevant observation nodes ............................................. 121
5.9 On the left is a strategy graph just after node $S_5$ has been added; on the right is the compressed strategy graph. .......................................................... 122
5.10 Nodes $S_4$ and $S_5$ have zero-probability branches that are not shown, allowing them to be merged. .............................................................. 122
5.11 The strategy graph created by eliminating decision variable $D_2$. ........ 123
5.12 The strategy graph created by eliminating observed chance variable $Z_2$. 124
5.13 The strategy graph created by eliminating decision variable $D_1$. ........ 124
5.14 The strategy graph created by eliminating observed chance variable $Z_1$, and it is also the final strategy graph for the demonstrate influence diagram. .... 125
5.15 Influence diagram for diagnosis/treatment problem based on the CHILD belief network, with utility table. ................................................................. 129
5.16 Optimal strategy graph for the Child influence diagram. .................... 132
5.17 Knee arthroplasty problem .............................................................. 133
5.18 Bounded-suboptimal strategy graph for Arthronet influence diagram. .... 136
CHAPTER 1

INTRODUCTION

1.1 Motivation

An influence diagram is a graphical model for representing problems of decision analysis, especially problems of decision making under imperfect information. Real-world applications of influence diagrams include medical decision making [39, 44], software risk analysis [12], software change management [7], real options problems [32], fisheries management [31], and many others. A related model for sequential decision modeling under uncertainty is a partially observable Markov decision process (POMDP) [56]. It is widely used in artificial intelligence and related fields of engineering, with applications to autonomous robot control, behavioral ecology, structural inspection, machine vision, network troubleshooting, moving target search, and more [11]. Although these two models are closely related and can represent the same class of problems, they have been developed by different research communities, and the algorithms developed to solve them are very different.

Traditional influence diagram algorithms explicitly or implicitly build a decision tree that includes all possible sequences of observations and decisions, and evaluate the decision tree backwards using dynamic programming to get the optimal expected value and the strategy associated with it. The overall time complexity of this approach is bounded by
the size of the decision tree, which is exponential in the number of relevant decision and observed variables. Since each branch in the tree corresponds to a distinct history, we call algorithms that use this approach history-based algorithms. History-based algorithms face two issues. The first issue is that the complexity of the algorithms is exponential in the length of the sequence of decisions and observations. The second issue is that the size of the representation of a strategy is also exponential in the length of the sequence of decisions and observations, which can make the resulting strategies difficult to interpret. My research will address those two difficulties and provide solutions that significantly improve the state of the art.

1.2 Contributions

This dissertation has two main contributions. First, we proposed an approach that integrates influence diagrams’ and POMPDs’ evaluation techniques to take advantages of both techniques, which dramatically improves the efficiency of the evaluation. Our experimental results show that for some influence diagrams our new approach has ten order of magnitude of speedup over the traditional influence diagram evaluation algorithms. Second, we proposed an approach that constructing and representing an optimal strategy compactly as a strategy graph using the generalized variable elimination algorithm.

**Integrating Influence Diagrams and POMDPs**

For the related model of finite-horizon POMDPs, a strategy and value function are often represented differently; at each stage, a strategy is represented as a mapping from belief states to the choice of action, where a belief state is a vector of posterior probabilities
over a state space, and the value function is represented as a set of real number vectors that contains optimal expected value information for all possible belief states. We call this approach a belief-based approach. Whereas the complexity of history based algorithms for solving IDs is exponential in the number of histories, the complexity of algorithms for solving POMDPs is independent of the number of histories; instead, it depends on the number of undominated strategies, or, more simply, it depends on the complexity of the strategies. In some cases, this alternative representation allows problems to be solved much more efficiently. Moreover, for problems with a horizon that is longer than a couple of stages, the belief-based representation of a strategy can be more compact, and sometimes much more compact, than a history-based representation.

We introduce two new algorithms that leverage this alternative representation of a strategy, and the related approach to dynamic programming originally developed for solving POMDPs. The first algorithm, called Generalized Arc-Reversal, is an extension of the classic node-removal/arc-reversal algorithm for ID evaluation. The second algorithm is a similar generalization of the variable elimination approach to solving IDs, called Generalized Variable Elimination. The variable elimination approach to ID evaluation is more widely-used than the node-removal/arc-reversal approach, and has several advantages. Among them, it uses potentials instead of conditional probabilities, which lets it avoid extra divisions incurred by arc reversals, and it allows variables to be eliminated in a less constrained order than nodes can be removed in the node reduction approach. These algorithmic differences lead to several differences between the generalization of the variable elimination approach compared to the generalization of the Arc reversal approach. Interestingly, both
the traditional variable elimination algorithm for IDs and the classic dynamic programming approach to solving finite-horizon POMDPs can be viewed as special cases of the more general algorithm we develop.

The main idea of our new approach is that we allow elimination of decision variables and observed variables before removing unobserved chance variables, which is not allowed by traditional algorithms for ID evaluation. By using the concepts of a belief state and a piecewise linear and convex value function representation, which were developed for solving POMDPs, our new approach allows elimination orders that are not allowed by the traditional approach, which can dramatically improve the scalability. Our new approach adopts dynamic programming techniques used in solving POMDPs to eliminate decision and observed variables when there are unobserved chance variables that have not yet been eliminated.

Our preliminary results demonstrate substantial speedup, especially in solving complex, multistage problems. For example, our two new generalized algorithms can solve a maze navigation problem [42] in less than 1 minute, whereas the traditional variable elimination and Arc-Reversal algorithm have no hope of solving this problem at all, even if run for days. Because the generalized algorithms also allow nodes to be removed in the same order as they are removed by the traditional algorithms, they are also guaranteed to perform at least as well as the traditional algorithms. The approach we developed also revealed the close relationship between the two dynamic programming techniques that are used in IDs and POMDPs, and it opens the possibility of integrating other ID evaluation algorithms with POMDP evaluation algorithms.
Representing an Optimal Strategy Compactly

Traditional ID algorithms compute an optimal strategy, but little attention has been given to the question of how to represent a strategy compactly. The traditional approach is to store a strategy in large tables, or represent it by a decision tree. In both cases, the size of the strategy representation is exponential in the number of decision and observed variables.

A strategy often contains a great deal of redundancy and structure that allow it to be compressed into a simpler and easier to understand form. We have developed an approach to representing a strategy more compactly as a graph, called a strategy graph. We remove unnecessary information from the graph by removing unreachable branches in the strategy tree, and we remove duplicate information by merging isomorphic subgraphs.

We show our approach to strategy representation and compression can be integrated in the generalized variable elimination algorithm, and the overhead is negligible. Our preliminary results show that the resulting strategy graph has significant and often dramatic compression compared to the decision tree representation [36], which is the current state of the art.

There is an interesting relationship between the strategy compression technique and the value function representation used in the POMDPs literature. If two strategies that appear in different histories are identical, which means that they can be compressed, it also means their expected value belongs to a single vector if we use the POMDP value function representation. In other words, those two strategies will be compressed automatically using
the belief-based approach. Therefore, the strategy compression is closely related to our first contribution. (In fact, this idea was inspired by work on the first contribution.)

Representing a strategy compactly as a graph makes it easier to understand by a human. In many fields, including medical decision making and financial investment, the interpretability of the strategy is very important because people are unlikely to accept and implement a strategy unless they understand it.

1.3 Dissertation Outline

This Dissertation is organized like follow. Chapter 2, we introduce the background of influence diagrams and POMDPs. Chapter 3, we introduce a new algorithm that integrating influence diagrams’ and POMDPs’ technique, called generalized variable elimination algorithm. We also introduce a heuristic of dynamically choosing variables elimination ordering and a way of apply bounded-error approximation for the generalized variable elimination algorithm. Then, we show the experimental results of it. Chapter 4, we introduce another new algorithm we developed called generalized arc-reversal algorithm. Chapter 5, we introduce how to construct and represent optimal strategy compactly as the strategy graph using generalized variable elimination algorithm. Chapter 6, conclusion.
In this chapter, we discuss backgrounds of influence diagrams, including how to represent an influence diagram and how to evaluate an influence diagram. After that, we also discuss backgrounds of partially observable Markov decision processes.

2.1 Influence Diagrams

Howard and Matheson [24] first proposed influence diagrams (IDs) as a compact representation of decision trees. Both influence diagrams and decision trees are used to represent and solve sequential decision problems. However, the size of a decision tree is exponential in the number of variables, while the size of an ID is quadratic in the number of variables. An ID also shows the relationship between problem variables more clearly. An ID can also be viewed as a Bayesian network that is extended with decision variables and utility functions.

2.1.1 Model Descriptions and Notations

Formally an ID can be defined as a combination of graphical part and numerical part. First, graphical part of an ID is defined on a directed acyclic graph with three kinds of nodes and edges. Chance nodes, drawn as circles, represent random variables, $X =$
\{X_1, \ldots, X_m\}$, as in a Bayesian network [45]. An arrow going into a chance node is a conditioning edge that represents conditional dependence. \textit{Decision nodes}, drawn as rectangles, represent controllable decision variables, $D = \{D_1, \ldots, D_n\}$. An arrow going into a decision node is an informational edge that indicates the agent observes the variable before it makes the decision. A dotted line arrow going into a decision node indicates the no-forgetting assumption that we will introduce in the later section. \textit{Value nodes}, which are drawn as diamonds and have no children, represent the preferences of the decision maker. The arrow going into a utility node indicates the utility function depends on this variable.

Second is Numerical part of an ID. Each random variable $X_i \in X$ is associated with a conditional probability table, $P_i = P(X_i|pa(X_i))$, where $pa(X_i) \subseteq X \cup D \backslash \{X_i\}$ denotes the set of parent variables of $X_i$ in the graph. Figure 2.3 shows the CPTs for the used car problem. Similarly, each decision variable $D_k \in D$ has a parent set $pa(D_k) \subseteq X \cup D \backslash \{D_k\}$, denoting the variables whose values are observed before the decision is made. Each value node is associated with a value function $V_i \in V = \{V_1, \ldots, V_q\}$ that assigns a scalar value to each instantiation of $pa(V_i) \subseteq X \cup D$. Given multiple value nodes, we assume the total value is their sum. Figure 2.4 shows the utility function for the used car problem.

\textbf{Regularity and No-forgetting Assumption}

An influence diagram is regular if there is a temporal ordering of the decision nodes, denoted $D_1, D_2, \ldots, D_n$. In other words, there is a sequence of decisions that the agent
Figure 2.1

Partial ordering in an influence diagram

needs to make. We can partition an ID into a collection of disjoint sets of chance nodes, denoted $I = I_0, I_1, ..., I_n$. Using $\succ$ to represent the partial order, we have

$$I_0 \succ D_1 \succ I_1 \succ ... \succ D_n \succ I_n,$$

(2.1)

where $I_0$ is a set of chance nodes observed before the first decision, $I_k$ is a set of chance nodes observed between $D_k$ and $D_{k+1}$, and $I_n$ is a set of chance nodes never observed [25]. Figure 2.1 shows an influence diagram has three decision nodes. The nodes labeled $I_3$ are unobserved chance nodes. The nodes labeled $I_1$ are observed before $D_2$ and $D_3$ but not $D_1$. In a regular influence diagram, we also assume no-forgetting, which means the agent will remember all the information that has been observed. In other words, all the arcs going into $D_k$ will also go into $D_{k+1}$, for all $k$, where $1 \leq k \leq n$. The dotted lines in Figure 2.1 are indicating the no-forgetting assumption.
2.1.2 Used Car Buyer Example

Figure 2.2 - Figure 2.4 show an example of an ID for the classic used car buyer problem [23, 48, 55]. In this example, Jin wants to buy a used car the condition of the vehicle is unknown. The used car has two possible conditions in which “Peach” or “Lemon”. There is a total of ten major subsystems in the car. The car is considered “Peach” if there is only one subsystem has a defect and is considered “Lemon” if there are six subsystems have defects. Jin also knows that there is 80% probability that the used car is a “Peach”, and 20% probability that the used car is a “Lemon”. Finally, Jin knows that to repair the “Peach” car will cost $40 and repair the “Lemon” car will cost $200.

![Influence Diagram for Used Car Buyer Problem](image)

Jin can buy an “anti-lemon guarantee” that is offered by the dealer for an additional $60. If the car is a lemon, the anti-lemon guarantee will cover the full repair cost; otherwise, the anti-lemon guarantee will cover half of the repair cost. Jin also has the choice to do...
some mechanical examinations that will help him figure out the condition of the used car. There are three different kinds of mechanical examinations with different costs. The first choice is to do the steering subsystem test alone with a price of $9. The second choice is to do both fuel and electrical subsystem test together, and the total price for both tests is $13. The third choice is to do a two-test sequence in which the price of the first test, test the transmission subsystem, is $10. After that, Jin can decide whether to test the differential subsystem with additional $4 based on the first test result. If there exists a defect in the subsystem that is being tested, the test is guaranteed to detect it. Figure 2.2 shows the utility function table for all conditions.

The first test decision is represented as the decision variable Test1 in Figure 2.2. The Test1 has four options: “no test, testing the steering subsystem alone, testing the fuel and
<table>
<thead>
<tr>
<th>Test1</th>
<th>No Test</th>
<th>Fuel/Elect</th>
<th>Differential</th>
</tr>
</thead>
<tbody>
<tr>
<td>Test2</td>
<td>No Test</td>
<td>Differential</td>
<td>Fuji/Electric</td>
</tr>
<tr>
<td>Buy</td>
<td>Not Buy</td>
<td>Buy</td>
<td>Buy Guaranteed</td>
</tr>
<tr>
<td>Condition</td>
<td>P</td>
<td>L</td>
<td>P</td>
</tr>
<tr>
<td>Profit</td>
<td>0</td>
<td>0</td>
<td>60</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Test1</th>
<th>No Test</th>
<th>Steering</th>
<th>Differential</th>
</tr>
</thead>
<tbody>
<tr>
<td>Test2</td>
<td>No Test</td>
<td>Differential</td>
<td>Steering</td>
</tr>
<tr>
<td>Buy</td>
<td>Not Buy</td>
<td>Buy</td>
<td>Buy Guaranteed</td>
</tr>
<tr>
<td>Condition</td>
<td>P</td>
<td>L</td>
<td>P</td>
</tr>
<tr>
<td>Profit</td>
<td>-13</td>
<td>-13</td>
<td>47</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Test1</th>
<th>No Test</th>
<th>Transmission</th>
<th>Differential</th>
</tr>
</thead>
<tbody>
<tr>
<td>Test2</td>
<td>No Test</td>
<td>Differential</td>
<td>Transmission</td>
</tr>
<tr>
<td>Buy</td>
<td>Not Buy</td>
<td>Buy</td>
<td>Buy Guaranteed</td>
</tr>
<tr>
<td>Condition</td>
<td>P</td>
<td>L</td>
<td>P</td>
</tr>
<tr>
<td>Profit</td>
<td>-10</td>
<td>-10</td>
<td>50</td>
</tr>
</tbody>
</table>

Figure 2.4

The Utility Table for Used Car Buyer Problem
electrical subsystems, and testing the transmission subsystem with a possibility of testing the differential subsystem next”. The first test results are represented as the random variable \( \text{Result1} \) in Figure 2.2. The possible results for \( \text{Result1} \) are “no result, no defect, one defect, and two defects”. The probability distribution of the variable, conditioned on \( \text{Test1} \) and \( \text{Condition} \) is given in Figure 2.3.

The second test decision is represented as the decision variable \( \text{Test2} \) in Figure 2.2. \( \text{Test2} \) has two choices: “no test and testing the differential subsystem”. The second test results are represented as The random variable \( \text{Result2} \). \( \text{Result2} \) has three possible results, which are “no result(nr), no defect(zero) and one defect(one)”. The probability distribution of the variables, conditioned on \( \text{Test1}, \text{Result1}, \text{Test2} \) and \( \text{Condition} \), is given in Figure 2.4.

The decision variable \( \text{Buy} \) represents the purchase decision. The frame for \( \text{Buy} \) has three elements: “not buying the car, buying the car without the anti-lemon guarantee and buying the car with the anti-lemon guarantee”.

The goal for this problem is to find an optimal strategy based on the initial belief of the car condition that maximizes the buyer’s profit, where the strategy specifies which tests to perform, and whether to buy the car as a function of the information available to the buyer. Although the condition of the used car is unobservable, the tests provide imperfect information about the condition of the used car at different costs. The optimal strategy for this problem is to perform the fuel/elect test. If the test result is no defects, then the decision is to buy the car without also purchasing a warranty; otherwise, the decision is to buy the car and a warranty. The maximum expected utility is $32.87.
2.1.3 Variables Independency Analysis

Analysis the independencies between variables in influence diagrams are very important since it can be used to reduce the computation cost of IDs evaluation substantially. A key technique that is used to identify that is called d-separation analysis, and let us introduce the concepts of d-separation first.

D-separation

The concept of d-separation is first proposed in Bayesian networks (BNs). Analysis the d-separation property in the graph gives us a way that identifies independencies that hold in a distribution associated with a Bayesian network (BN). The same method can be also used to analysis the d-separation property for influence diagrams since IDs have the same graphical structure as BNs except that IDs have decision nodes and utility nodes in addition to the chance nodes. Identify d-separation between two nodes only depend on the graphical structure of the BNs or IDs, the numerical part of the BNs or IDs, and the types of the nodes do not matter. In IDs, decision node, and utility node are all considered evidence. Therefore, in order to make it clear and simple, we only introduce the definition and the approach to identify d-separation for BN. Identifying d-separation property between two nodes is also a subroutine that is used in finding the relative past variables for the influence diagram, which we will discuss later. [26, 30]

There are only three possible different kinds of connection that will occur in Bayesian networks and influence diagrams. They are serial connections, diverging connections, and converging connections, and we introduce them one by one as follow.
Figure 2.5
Serial Connections

Figure 2.6
Diverging Connections

Figure 2.7
Converging Connections
Serial Connections, Consider the situation shows in Figure 2.5. Here C is influenced by B and B is influenced by A. Therefore, the certainty of B is depended on the evidence of A and the certainty of C is also depended on the evidence of A if B is unknown. However, if the state of B is known then A and C become independent, and we call that A and C are d-separated given B. [26, 30]

Diverging Connections, The situation shows in Figure 2.6 is called a diverging connection. Influence can pass between all the children of A if A is unknown. On the other hand, if A is instantiated, then B, C, ..., E are d-separated. [26, 30]

Converging Connections, The situation shows in Figure 2.7 is called a converging connection. If A is unknown then B, C, ... E are independent, which means knowing one of them cannot influence the certainties of the others through A and we called B, C, ... E are d-separated. However, if A is instantiated, then B, C, ... E become dependent. [26, 30]. By knowing those three situations, we can define d-separation as follow:

**Definition 1 (d-separation)**

*Two distinct variables A and B, in a BN are d-separated if for all paths between A and B, there is an intermediate variable V (distinct from A and B) such that either the connection is serial or diverging and V is instantiated or the connection is converging, and neither V nor any of V’s descendants have received evidence.* [26, 30]

If A and B are not d-separated, we call them *d-connected.*
**Required Past Variables**

In general, both the computation cost for solving an influence diagram and the size of the optimal solution increase exponentially in the number of the *required past variables* for each of the decisions. For a given influence diagram, not all variables in the past for a decision are necessarily required when evaluating it. Therefore, we want to identify the non-required past variables to reduce the computation cost and also the size of the solution. All the existing algorithms for solving IDs have smartly designed to take the advantages of not do the computations on the non-required past variables at each step, such as arc-reversal algorithm and variable elimination algorithm. Let us give the formal definitions of required variables first.

**Definition 2 (Required variables)**

Let $I$ be an influence diagram and let $D$ be a decision variable in $I$. The variable $X \in \text{past}(D)$ is said to be required for $D$ if there exist a realization $R$ of $I$, a configuration $y$ over $\text{dom}(\delta_D) \setminus \{X\}$, and states $x_1$ and $x_2$ of $X$ such that $\delta_D(x_1, y) \neq \delta_D(x_2, y)$, where $\delta_D$ is an optimal policy for $D$ with respect to $R$. The set of variables required for $D$ is denoted by $\text{req}(D)$. [26]

We want to identify the required past variables for a decision variable $D$, first we identify the *relevant utility variable*, the definition of it shows in Definition 3, and Corollary 1 shows how to identify the relevant utility variables. Finally, Corollary 2 shows how to identify the required past variables.
Definition 3 (Relevant utility variables)

The utility function \( U \) is relevant for decision \( D \) if there exists two realizations \( R_1 \) and \( R_2 \) of \( I \) that differ only on \( U \) such that the optimal policies for \( D \) are different in \( R_1 \) and \( R_2 \). [26]

Corollary 1 (Identifying the relevant utility variables)

Let \( D_n \) be the last decision variable in the influence diagram \( I \), and let \( U \) be a utility node in \( I \). Then \( U \) is relevant for \( D_n \) if and only if there is a directed path from \( D_n \) to \( U \). [26]

Corollary 2 (Identifying the required past variable)

Let \( D \) be the last decision variable in the influence diagram \( I \) and let \( X \) be a variable in \( \text{past}(D) \). Then \( X \) is required for \( D \) if and only if \( X \) is d-connected to a utility node relevant for \( D \) given \( \text{past}(D) \setminus \{X\} \). [26]

2.2 Evaluating Influence Diagrams

The objective of evaluating influence diagram is to find a strategy that will lead to the maximum expected utility. Therefore, we are going to introduce strategy and the representation of strategy first, then introduce how to compute maximum expected utility.

2.2.1 Strategy and Strategy Representation

A strategy for an influence diagram is a list of decision rules \( \Delta = (\delta_1, \ldots, \delta_n) \), one for each decision variable \( D_i \in D \), where a decision rule is a mapping, \( \delta_i : \text{dom}(\text{pa}(D_i)) \rightarrow \text{dom}(D_i) \), that prescribes an action for each instantiation of the parent variables \( \text{pa}(D_i) \).

An ID is solved by computing a strategy with the maximum expected utility. Because the decision rule \( \delta_i \) for the decision variable \( D_i \) is conditioned on all possible instantiations of
the set $I_0 \cup D_1 \cup I_1 \ldots D_{i-1} \cup I_{i-1}$, the complexity of ID evaluation can grow exponentially in the length of the history. (In limited cases, structural analysis of the ID can distinguish relevant from irrelevant variables in this history, which can help slow this exponential growth in complexity [33, 41, 50].)

2.2.2 Probability Computations in Influence Diagrams

Compute probability is a key step that is used to compute expected utility in influence diagram. Therefore, we give some backgrounds of how to compute those probabilities in influence diagrams. The method for compute probability is adopted from Bayesian networks.

Bayes’s Rule

Bayes’s Theorem describes the probability of an event, based on prior knowledge of conditions that might be related to the event. For example, if we know the prior probability of $A$, denoted $P(A)$, the prior probability of $B$, denoted $P(B)$ and the conditional probability of $A$ depends on $B$, denoted $P(A|B)$, then we can compute the conditional probability $B$ depends on $A$, denoted $P(B|A)$ based on Bayes’s rule [26].

**Theorem 1 (Bayes’s rule)**

$$P(B|A) = \frac{P(A|B)P(B)}{P(A)} = \frac{P(A, B)}{\sum P(A, B)}$$ (2.2)

and conditioned on another variable $C$ we have:

$$P(B|A, C) = \frac{P(A|B, C)P(B|C)}{P(A|C)} = \frac{P(A, B|C)}{\sum P(A, B|C)}$$ (2.3)
**General Chain Rule**

Let $U = \{V_1, ..., V_n\}$ be a set of total variables. General chain rule is used to calculate the joint probability table $P(U) = P(V_1, ..., V_n)$.

**Corollary 3 (The general chain rule)**

Let $U = \{V_1, ..., V_n\}$ be a set of variables. The joint probability for any probability distribution $P(U)$ we have [26]:

$$P(U) = P(V_n|V_1, ...V_{n-1})P(V_{n-1}|V_1, ...V_{n-2})...P(V_2|V_1)P(V_1).$$  \hfill (2.4)

**Chain Rule for Influence Diagrams:**

For influence diagrams the chain rule is different compared to general chain rule. The reason for that is because influence diagrams have decision variables that are different from chance variables. Since we have full control of the decision, it requires no prior probability. Also, the probability distribution to a chance variable is only meaningful when a decision has been taken and the action performed [26].

**Theorem 2 (The chain rule for influence diagrams)**

Let $ID$ be an influence diagram with universe $U = U_C \cup U_D$ where $U_C$ is a set of chance variables and $U_D$ is a set of decision variables. Then [26]:

$$P(U_C|U_D) = \prod_{X \in U_C} P(X|pa(X)).$$  \hfill (2.5)
Let’s look at the influence diagram \textit{EID} in Figure 2.8. From the equation 2.5 we have:

\[
P(X_3, Z, X_2, X_1|D_1, D_2) = P(X_3|Z, X_2, X_1, D_1, D_2)P(Z, X_2, X_1|D_1, D_2) \\
= P(X_3|Z, X_2, X_1, D_1, D_2)P(Z|X_2, X_1, D_1, D_2) \\
\times P(X_2|X_1, D_1, D_2)P(X_1|D_1, D_2).
\]  

(2.6)

Since \(X_3\) is d-separated from \(X_1, Z, \) and \(D_1\) when \(X_2\) and \(D_2\) is given, we have:

\[
P(X_3|Z, X_2, X_1, D_1, D_2) = P(X_3|X_2, D_2).
\]

for the same reason we also have:

\[
P(Z|X_2, X_1, D_1, D_2) = P(Z|X_2, X_1), \\
P(X_2|X_1, D_1, D_2) = P(X_2|X_1), \\
P(X_1|D_1, D_2) = P(X_1|D_1).
\]

Substituting those equations into equation 2.6, we have:

\[
P(X_3, Z, X_2, X_1|D_1, D_2) = P(X_3|X_2, D_2)P(Z|X_2, X_1)P(X_2|X_1)P(X_1|D_1). \tag{2.7}
\]

which is the product of the probability potentials for Example \textit{EID}. 

21
2.2.3 Maximum Expected Utility

To solve an influence diagram, the basic idea is to unfold it into a decision tree explicitly and calculate the expected utility for each branch, then find the action that has the maximum expected utility. Figure 2.9 is the decision tree of influence diagram $EID$. The expected utility of performing action $d^1_2$ given $(d^1_1, z_1)$ is:

$$EU(d^1_2 | d^1_1, z_1) = \sum_{X_1, X_3} P(X_1, X_3 | d^1_1, d^1_2)(V_1(X_1, d^1_1) + V_2(X_3))$$

For the action $d^2_2$, we have:

$$EU(d^2_2 | d^1_1, z_1) = \sum_{X_1, X_3} P(X_1, X_3 | d^1_1, d^2_2)(V_1(X_1, d^2_1) + V_2(X_3))$$

Put those two equations together, we have:

$$EU(D^2_2 | d^1_1, z_1) = \sum_{X_1, X_3} P(X_1, X_3 | d^1_1, D_2)(V_1(X_1, D_1) + V_2(X_3))$$

We choose the action that has the maximum expected utility, and we get a decision rule for $D_2$ when $D_1 = d^1_1$ and $Z = z_1$:

$$MEU_{D_2}(d^1_1, z_1) = \max_{D_2} EU(D_2 | d^1_1, z_1).$$

$$\delta_{D_2}(d^1_1, z_1) = arg\max_{D_2} EU(D_2 | d^1_1, z_1). \quad (2.8)$$

which $MEU_{D_2}(d^1_1, z_1)$ is the maximum expected utility for branch $d_1$ and $z_1$, and $\delta_{D_2}(d^1_1, z_1)$ is the action leads to the maximum expected utility for this branch.
Figure 2.9

The decision tree of ID EID

23
Generalizing these two equations in Equation (2.8) to any path over $D_1, Z$, we get a new utility function and policy:

$$MEU_{D_2}(d_1, Z) = \max_{D_2} EU(D_2|D_1, Z)$$

$$\delta_{D_1}$$

Next, for the decision $D_1$, let us look at Figure 2.10. If we take the action $d_1$, we get the expected utility:

$$EU(d_1) = P(z_1|d_1) MEU_{D_2}(d_1, z_1) + P(z_2|d_1) MEU_{D_2}(d_1, z_2),$$
which can also be written:

\[ EU_{D_1} = \sum_z \hat{P}(Z|D_1) MEU_{D_2}(D_1, Z). \]

The max expect utility is:

\[ MEU_{D_1} = \max_{D_1} \sum_z \hat{P}(Z|D_1) MEU_{D_2}(D_1, Z) \quad (2.10) \]

The optimal decision is

\[ \delta_{D_1} = \arg\max_{D_1} \sum_z \hat{P}(Z|D_1) MEU_{D_2}(D_1, Z) \]

Combine Equation (2.9) and Equation (2.10) we get:

\[
MEU_{D_1} = \max_{D_1} \sum_z P(Z|D_1) \max_{D_2} \sum_{X_1, X_2, X_3} \left( P(X_1, X_3|D_1, Z, D_2)(V_1(X_1, D_1) + V_2(X_3)) \right) \\
= \max_{D_1} \sum_z \max_{D_2} \sum_{X_1, X_3} \left( P(Z|D_1) P(X_1, X_3|D_1, Z, D_2) V_{total} \right) \\
= \max_{D_1} \sum_z \max_{D_2} \sum_{X_1, X_3} \left( P(Z|D_1, D_2) P(X_1, X_3|D_1, Z, D_2) V_{total} \right) \\
= \max_{D_1} \sum_z \max_{D_2} \sum_{X_1, X_3} \left( P(X_1, X_3, Z|D_1, D_2) V_{total} \right) \\
= \max_{D_1} \sum_z \max_{D_2} \sum_{X_1, X_2, X_3} \left( P(X_1, X_2, X_3, Z|D_1, D_2) V_{total} \right) \\
= \max_{D_1} \sum_z \max_{D_2} \sum_{X_1, X_2, X_3} \left( P(U_C|U_D) V_{total}, \right)
\]

and for \( \delta_{D_1} \):

\[
\delta_{D_1} = \arg\max_{D_1} \sum_{D_2} \sum_{X_1, X_2, X_3} \left( P(U_C|U_D) V_{total}. \right)
\]
Theorem 3 (Maximum expect utility and optimal strategy for decision $D_i$)

Let $ID$ be an influence diagram over $U = UC \cup UD$ and $U_V = V_i$. Let the temporal order of the variables be described as $I_0 \succ D_1 \succ I_1 \succ ... \succ D_n \succ I_n$, and let $V = \sum_i V_i$. Then [26]:

(i) An optimal policy for $D_i$ is

$$\delta_i(I_0, D_1, ..., I_{i-1}) = \arg\max_{D_i} \sum_{I_i} \max_{D_{i+1}} \ldots \max_{D_n} \sum_{I_n} P(U|U_D) V.$$  \hspace{1cm} (2.11)

(ii) The expected utility from the policy $\delta_i$ is

$$MEU_i(I_0, D_1, ..., I_{i-1}) = \frac{1}{P(I_0, ..., I_{i-1}|D_1, ..., D_{i-1})} \max_{D_i} \sum_{I_i} \max_{D_{i+1}} \ldots \max_{D_n} \sum_{I_n} P(U|U_D) V.$$  \hspace{1cm} (2.12)

and the strategy for ID consisting of an optimal policy for each decision yields the maximum expected utility:

$$MEU(ID) = \sum_{I_0} \max_{D_1} \sum_{I_1} \ldots \max_{D_n} \sum_{I_n} P(U|U_D) V.$$  \hspace{1cm} (2.13)

2.2.4 Classic Algorithms

Algorithms for solving IDs can be classified as indirect and direct methods. Indirect methods convert an ID to a secondary structure before solving it, which can be a decision tree that is solved by backwards induction or branch-and-bound search [24, 47, 60], a belief network that is solved by probabilistic inference techniques [14, 52, 62], or a junction tree that is solved by message-passing techniques [25]. Direct methods solve an ID in the course of performing a sequence of value-preserving transformations on the ID itself,
which may involve removing one node at a time from the network [43, 51, 59], or one variable at a time from an equivalent mathematical formula [15].

Shachter [51] describes an algorithm that directly solves an ID without first transforming it into decision tree. Shachter’s algorithm removes one node from the ID each iteration until the only node left is the utility node. The algorithm is called the Arc Reversal algorithm because it includes an important step called arc reversal. Tatman and Shachter [59] improved the Arc Reversal algorithm by allowing multiple utility nodes that are combined using a “super value node”. Jensen et al. [25] describe a way to solve an ID by transforming it into a junction tree and then applying a message passing algorithm. Dechter [15] proposes a bucket elimination algorithm.

Algorithms for solving IDs [25, 51, 59] are closely related to algorithms for solving decision trees in that they consider every possible sequence of decisions and observations, beginning from an initial prior probability distribution over unobserved states; that is, they consider all possible histories. Probabilities are updated in a forwards direction by Bayesian conditioning, and the optimization problem is solved in a backwards direction by a dynamic programming approach that considers decisions in reverse order.

2.3 Partially Observable Markov Decision Processes

Partially observable Markov decision processes (POMDPs) were first studied in the control theory and operations research communities [1, 16, 38, 54, 56, 57]. Over the past twenty years, the artificial intelligence community has adopted POMDPs as a framework for planning under uncertainty [5, 28]. A POMDP is an extension of a completely ob-
servable MDP that allows imperfect or incomplete information about the current state of a system. With this generalization, POMDPs provide a natural framework for modeling problems with partial observability and uncertain action effects.

2.3.1 Model Descriptions and Notations

A POMDP is specified by a tuple \( \langle S, A, Z, T, O, r, H \rangle \), where: \( S \) is a finite set of states of a system; \( A \) is a finite set of actions that an agent can execute; \( Z \) is a finite set of observations that an agent can observe; \( T : S \times A \times S \rightarrow [0, 1] \) is a state transition function that maps \( S \times A \) into discrete probability distributions over \( S \), where \( Pr(s'|s, a) \) denotes the probability that taking action \( a \in A \) in state \( s \in S \) results in a transition to state \( s' \in S \); \( O : A \times S \times Z \rightarrow [0, 1] \) is an observation function that maps \( S \times A \) into discrete probability distributions over \( Z \), where \( Pr(z|s', a) \) denoted the probability that \( z \in Z \) is observed after
taking action \( a \in A \) and making a transition to state \( s' \in S \); \( r : S \times A \rightarrow \mathbb{R} \) is a reward function that maps \( S \times A \) into real numbers that represent expected rewards where \( r(s, a) \) denotes the immediate reward for taking action \( a \in A \) in state \( s \in S \); and \( H \) denotes a horizon number which is the number of time-steps the agent must plan for. The horizon can be finite or infinite.

A finite-horizon POMDP can be viewed as a special case of an ID, defined as follows. For each stage \( t \) in a finite sequence of stages, there is an unobserved state variable \( S_t \), an observation variable \( Z_t \), a decision variable \( D_t \), and a reward function \( V_t \). The state and observation variables correspond to the chance nodes of an ID, the decision variables correspond to the decision nodes, and the reward functions correspond to the value nodes, as shown in Figure 2.11. The state variables at each stage satisfy the *Markov property*, which means that the state at stage \( t \) depends only on the state and action at the previous stage, and the observation and reward at stage \( t \) depend only on the state at stage \( t \) and the previous state and action.

### 2.3.2 POMDP Strategy Representation

Let’s use \( \Pi_h \) to denote the set of all policies \( \pi(s) \) at horizon \( h \). For completely observable MDPs, the policy is a mapping from state to action since the system always satisfies the Markov property. However, for POMDPs the Markov property no longer holds because the agent is not certain which state it is in. Therefore, the agent has to remember the histories that agent has made and observed. In POMDPs literature, there is a way that could
remember the history information using “belief state”. Let us introduce “belief state” as follows:

A belief state $b_t$ is a probability distribution over $S$, let $b_t(s)$ denote the probability assigned to state $s$ by belief state $b_t$. Using Bayes theorem, we can compute belief state $b_t$ from the previous belief state $b_{t-1}$, the previous action $a_t$ and the current observation $z_t$:

$$b_t(s') = \frac{Pr(z_t|a_{t-1}, s') \sum_{s' \in S} Pr(s'|s, a_{t-1}) b_{t-1}(s)}{\sum_{s' \in S} [Pr(z_t|a_{t-1}, s')] \sum_{s \in S} Pr(s'|s, a_{t-1}) b_{t-1}(s)}$$

(2.14)

A belief state $b_t$ is a sufficient statistic that includes all relevant information about history. In other words, belief state satisfies the Markov property and knowing belief state $b_t$, all the histories before no longer matter [1].

Therefore, there is a mapping from a belief state to a strategy based on actions and observations. More precisely, when the number of actions and observations is finite, the set $\Gamma_t$ of all possible $t$-stage strategies can be defined recursively, as follows:

$$\Gamma_1 = D,$$

(2.15)

$$\Gamma_t = \{ (d_t, \gamma_t) | d_t \in D, \gamma_t : Z \rightarrow \Gamma_{t-1} \}, \text{ for } t > 1,$$

(2.16)

where $\gamma_t^i \in \Gamma_t$ denotes a specific $t$-stage strategy indexed by $i$. A $t$-stage strategy $\gamma_t^i$ is defined as a tuple $(d_t^i, \gamma_t^i)$ consisting of an action $d_t^i \in D$ and a mapping $\gamma_t^i : Z \rightarrow \Gamma_{t-1}$, such that each observation $z \in Z$ is mapped to a $(t - 1)$-stage strategy.

### 2.3.3 Value Function Representation

A very important result that Smallwood and Sondik proved is that the optimal value function is piecewise linear and convex for finite-horizon POMDPs [54], and can be ap-
proximated arbitrarily closely by a piecewise linear and convex function for infinite-horizon POMDPs [57]. Therefore, instead of using a single scalar value to represent the value function for a single belief state, POMDPs use a real-valued $|S|$-dimensional vector where $S$ is given a state set, and each value $\gamma : S \to \mathbb{R}$. For this state set, a belief state $b : S \to \mathbb{R}$ is also a real-valued $|S|$-dimensional vector, where the values are non-negative and sum to 1, since they represent a probability distribution. The value of the strategy for a given belief state $b$ is defined as

\[
V_\gamma(b) = \sum_{s \in S} b(s)\gamma(s). 
\]  

(2.17)

Note that, Equation (2.17) shows that this $|S|$-dimensional vector is a continue value function that can represent value for any belief states over $S$. Moreover, from Equation (2.16)
we knew a strategy is a mapping from a belief state to a strategy. Therefore, there is a mapping from each $|S|$-dimensional vector to a strategy [54, 57].

Given a set of possible strategies, $\Gamma$, each with a corresponding $|S|$-dimensional vector, the optimal strategy for a given belief state is determined by finding the strategy that maximizes the expected utility for this belief state, defined as

$$V(b) = \max_{\gamma \in \Gamma} \sum_{s \in S} (s)\gamma(s).$$  \hspace{1cm} (2.18)

From equation (2.18) we know the value for a given belief state $b(s)$ is the upper surface that is constructed by all vectors. For example in Figure 2.12, the value function is constructed by vector 1, 3, and 4. One observation we noticed here is that some of the vectors are dominated by others. For example in Figure 2.12 vector 2 and 5 are dominated by the combination of vector 1, 3, and 4. Removing those dominated vectors won’t affect the optimally expect utility for the system [3]. We call the process of removing dominated vectors pruning, and we will discuss how to prune dominated vectors in next section.

### 2.3.4 Eliminating Dominated Vectors

For every piecewise linear and convex value function, there is a minimal set of vectors that represents it [35]. Given a set $\Gamma_t$ of $t$-stage strategies with a corresponding set of value vectors $\mathcal{V}_t = \{v^i_t\}_{i = 1, \ldots, |\Gamma_t|}$, a strategy $\gamma^i_t \in \Gamma_t$ is said to be dominated by the other strategies $\Gamma_t \setminus \{\gamma^i_t\}$ if for all belief states $b$:

$$\sum_{x \in X} b(x)v^i_t(x) \leq \max_{v^k_t \in \mathcal{V}_t \setminus \{v^i_t\}} \sum_{x \in X} b(x)v^k_t(x).$$  \hspace{1cm} (2.19)

The condition given by (2.19) can be tested by solving a linear program. For a given vector $v$, the linear program shows in Algorithm 1 tests whether adding it to a set of vectors
\textbf{Algorithm 1: }Linear programing for testing vector dominance

\textbf{Input:} A vector \(v\) and a set of vectors \(\mathcal{V}\)

\textbf{Output:} \(\{d, b(s)\}\), \(d\) is real number that indicate the maximum value increase by adding \(v\) into \(\mathcal{V}\). \(b(s)\) is the belief state that leads to the maximum value increase.

1. Linear program LP setup:

2. Variables: \(d, b(s)\) for all \(s \in S, v, \mathcal{V}\)

3. Maximize: \(d\)

4. Constraints: 
   \[b \times v - b \times v_k - d \geq 0\] for all \(v_k \in \mathcal{V}\backslash\{v\}\)
   \[\sum_{s \in S} b(s) = 1\]
   \[b(s) \geq 0\] for all \(s \in S\)

5. Solve the linear program LP.

6. \textbf{return:} \(\{d, b(s)\}\)

\(\mathcal{V}\) (which does not include \(v\)) improves the value function represented by \(\mathcal{V}\) for any belief state. If it does, the improved value \(d > 0\), if not \(d \leq 0\), and means \(v\) is a dominated vector and we can eliminate it.

A naive approach for pruning a set of vectors \(\tilde{\mathcal{V}}\) to yield a minimal set \(\mathcal{V}\) is first to pruning any vector that is \textit{pointwise dominated} [17] by another. Pointwise dominance is a special case of dominance. A vector \(v\) is pointwise dominated by another vector \(v'\), if for each state \(s\), \(v(s) \leq v'(s)\). Then test every \(v \in \tilde{\mathcal{V}}\) using linear program shows in Algorithm 1. If \(v\) is dominated \((d \leq 0)\) by other vectors in \(\tilde{\mathcal{V}}\) then remove it from \(\tilde{\mathcal{V}}\) until there is no \(v\) can be removed.

A more efficient pruning technique is due to Lark [11]. Lark’s approach instead of gradually pruning \(\tilde{\mathcal{V}}\) until it is a minimal set, it maintains two sets of vectors, \(\hat{\mathcal{V}}\) and \(\mathcal{V}\), and gradually builds \(\mathcal{V}\) by adding to it vectors from \(\hat{\mathcal{V}}\) that are not dominated, shows in Algorithm 2. For the rest of this dissertation, when we call prune function, we always mean to use Lark’s approach.
Algorithm 2: Lark’s algorithm for pruning a set of vectors

**Input:** A non-minimal set of vectors $\tilde{V}$ representing a value function

**Output:** A minimal set of vectors $V$ representing the same value function

1. $V \leftarrow \emptyset$
2. Prue all pointwise dominated vectors from $\tilde{V}$
3. while $\tilde{V} \neq \emptyset$ do
   4. Pick one $v \in \tilde{V}$
   5. $\{d, b(s)\} \leftarrow \text{LP}(v, \tilde{V})$
   6. if $d > 0$ then
      7. $v^* \leftarrow \max_{v \in \tilde{V}} (v \cdot b(s))$
      8. remove $v^*$ from $\tilde{V}$
      9. add $v^*$ into $V$
   10. else
      11. remove $v$ from $\tilde{V}$
   12. end
4. return $\{A \text{ minimal set of vectors } V\}$

2.3.5 Evaluating POMDPs

For the past couple of decades, researchers have proposed many algorithms to solve POMDPs, including Sondik’s One Pass algorithm [56], Monahan’s Algorithm [38], Cheng’s Linear Support algorithm [13], the Witness Algorithm [3], Incremental Pruning [10], and region-based incremental pruning [19]. These are exact algorithms that belong to the value iteration approach. Sondik’s [57] and Hansen’s [20] policy iteration algorithms are exact algorithms that belong to the policy iteration approach. Pineau et al.’s [46] point-based value iteration is an approximation algorithm.

A POMDP can be transferred into a MDP with belief states, and value function that is defined in (2.18). Therefore, the dynamic programming, such as value iteration, can be used for evaluating POMDPs. When solving a POMDP, intuitively, the value function
$V_t(b)$ of stage $t$ can be computed from the value function $V_{t-1}(b)$ of stage $t-1$. This idea can be formalized by Bellman’s equation:

$$V_t(b) = \max_{a \in A} \sum_{s \in S} ((s, a)b(s) + \sum_{z \in Z} \sum_{s \in S} b(s)Pr(z|s, a) V_{t-1}(b'))$$ \hfill (2.20)

where $\sum_{s \in S} r(s, a)b(s)$ is the immediate reward for taking action $a$, $\sum_{s \in S} b(s)Pr(z|s, a)$ is the probability of observing $z$ when taking action $a$ in belief state $b$, and $V_{t-1}(b')$ is a result of Equation (2.18) where $b'$ is the result of Equation (2.14) by taking action $a$ and receive observed $z$. Although algorithms used in ID literature and algorithms used in POMDPs use the same Bellman equation, POMDP algorithms do not request building a history tree first and then solving it backward from the leaves. The backup is simply done backwards since we represent value functions as vectors that contain value information for all possible belief states.

### 2.3.6 Incremental Pruning Algorithm

Incremental pruning algorithm [10] is an exact algorithm using value iteration approach. The equation 2.20 can be break into 3 parts without effect the optimal value function as below:

$$V_{a,z}^t(b) = \sum_{s \in S} \frac{r(s, a)b(s)}{|Z|} + r \sum_{s \in S} b(s)Pr(z|s, a) V_{t-1}(b')$$ \hfill (2.21)

$$V_a^t(b) = \oplus_{z \in Z} V_{a,z}^z(b)$$ \hfill (2.22)

$$V^t(b) = \max_{a \in A} V_a^t(b)$$ \hfill (2.23)

$V_{a,z}^t(b)$ is the updated value function by taking an action $a$ and receive an observation $z$ based on one previous value function. This step is referred as the backprojection step.
of the algorithm. We use \( \mathcal{V}_{a,z}^t \) to denote the set of vectors generate by each action \( a \) and observation \( z \).

\( V_a^t(b) \) is the updated value function for execute action \( a \) by summing up all possible combination \( V_{a,z}^t(b) \), this step is called cross-sum step, where the \textit{cross sum} of two sets of vectors \( A \) and \( B \) is defined as \( A \oplus B = \{ a + b | a \in A, b \in B \} \). We use \( \mathcal{V}_{a}^t \) denote the set of vectors generate by each action \( a \). The algorithm is called \textit{incremental pruning} when the cross-sum step is performed efficiently by pruning as follows:

\[
\mathcal{V}_{a}^t = PR(\ldots PR(PR(V_{a,z}^{t,1} \oplus V_{a,z}^{t,2}) \oplus V_{a,z}^{t,3}) \ldots \oplus V_{a,z}^{t,k}).
\] (2.24)

The goal of this increment prune step shows in Equation (2.24) is to prune dominated vectors as early as possible.

\( V^t(b) \) is the optimal value function for horizon \( t \) by maximum \( V_a^t(b) \), We use \( \mathcal{V}^t \) denote the set of vectors generate by horizon \( t \). This step is referred to as the \textit{maximization} step.

The pseudocode for the incremental pruning algorithm is show in table algorithm 1.

Theoretically the number of vector grows double exponentially without prune dominated vectors by:

\[
\mathcal{V}_t = |A||\mathcal{V}_{t-1}|^{|Z|}
\] (2.25)

Fortunately for most of the problems, only need a small part of the vectors to maintain the optimal value function. The way incremental pruning algorithm doing this is applying pruning as early as possible.
Algorithm 3: Incremental pruning algorithm

1. Input: A set of vectors $\mathcal{V}^{t-1}$ that represents value function for stage $t - 1$
2. **foreach action $a$ and observation $z$ pair do**
3. 3. Generate $\mathcal{V}_{a,z}^t$ using equation (2.21)
4. 4. $\mathcal{V}_{a,z}^t \leftarrow$ Prune ($\overline{\mathcal{V}_{a,z}^t}$)
5. **end**
6. **foreach action $a$ do**
7. 7. $\mathcal{V}_a^t = \emptyset$
8. 8. **foreach observation $z$ do**
9. 9. $\overline{\mathcal{V}_a^t} = \mathcal{V}_a^t \oplus \mathcal{V}_{a,z}^t$
10. 10. $\mathcal{V}_a^t \leftarrow$ Prune ($\overline{\mathcal{V}_a^t}$)
11. **end**
12. **end**
13. Generate $\overline{\mathcal{V}^t}$ by $\bigcup_{a \in A} \Gamma_a^t$
14. $\mathcal{V}^t \leftarrow$ Prune ($\overline{\mathcal{V}^t}$)
15. Output: A set of vectors $\mathcal{V}^t$ that represents value function $V^t$
CHAPTER 3
THE GENERALIZED VARIABLE ELIMINATION ALGORITHM

In this chapter, we generalize the classic Variable Elimination algorithm for influence diagrams by integrating it with the dynamic programming approach to solving POMDPs reviewed in Section 2.3. In Chapter 4, we describe a similar generalization of the Arc Reversal algorithm for solving influence diagrams.

3.1 Traditional Variable Elimination Algorithm

We begin with a review of the traditional variable elimination algorithm for solving IDs, which is an extension of the well-known variable elimination approach to solving Bayesian networks [15, 26, 30, 37].

The variable elimination algorithm for IDs solves Equation (2.13) by progressively eliminating variables and replacing probability and utility functions that mention these variables with equivalent functions that do not. In contrast to the VE algorithm for Bayesian networks, there are restrictions on the order in which variables can be eliminated when solving an ID. In particular, variables must be eliminated in reverse order of the partial ordering imposed by the information constraints of the problem, which is called a strong elimination ordering [25].
A strong elimination order depends, first of all, on the ordering of decisions: $D_1, \ldots, D_n$. Given this ordering of decisions, the set of chance nodes, $V_C$, is partitioned according to when they are observed. $I_0$ is the set of chance nodes observed before decision $D_1$. $I_i$ is the set of chance nodes observed after $D_i$ but before $D_{i+1}$. $I_n$ are the chance nodes observed after the last decision $D_n$; in other words, they are not observed at all. Thus we have the following partial order among the nodes: $I_0 < D_1 < \ldots < D_n < I_n$. The variable elimination algorithm eliminates variables from the ID according to this partial order. When all variables have been eliminated, the maximum expected utility (MEU) has been found, together with an optimal strategy, as explained below.

3.1.1 Potentials and Operations on Potentials

The probability and utility functions manipulated by the VE algorithm are called “potentials” because they are not necessarily normalized. A probability potential over a set of variables $X$, denoted $\phi(X)$, is a non-negative real-valued mapping $\phi : sp(X) \to \mathbb{R}^{\geq 0}$.

A utility potential over a set of variables $X$, denoted $\psi(X)$, is a real-valued mapping $\psi : sp(X) \to \mathbb{R}$. The domain of a potential is denoted by $\text{dom}(\phi)$. For example, the domain of the potential $\phi(A, B, C)$ is $\text{dom}(\phi(A, B, C)) = \{A, B, C\}$.

The VE algorithm performs two kinds of operations on potentials: combination and marginalization. The combination of two or more potentials, whether probability or utility potentials, results in a new potential whose domain is the union of the combined potentials. Marginalization (which is sometimes called projection) results in the removal of a variable from the domain of a potential.
The following combination operations are used in the VE algorithm for IDs. Two or more utility potentials can be added by performing element-wise additions. Consider two potentials: \( \psi(X) \) and \( \psi'(Y) \). Their sum, \( \psi''(X, Y) = \psi(X) + \psi'(Y) \), is defined so that:

\[
\psi''(x, y) := \psi(x) + \psi'(y), \quad \text{for each } x, y \text{ of } X \cup Y. \tag{3.1}
\]

Two or more probability potentials can be multiplied by performing element-wise products. That is, \( \phi''(X, Y) = \phi(X) \cdot \phi'(Y) \) is defined so that:

\[
\phi''(x, y) := \phi(x) \cdot \phi'(y), \quad \text{for each } x, y \text{ of } X \cup Y. \tag{3.2}
\]

A probability potential and a utility potential can be multiplied by performing element-wise products, which creates a new utility potential. That is, \( \psi''(X, Y) = \phi(X) \cdot \psi'(Y) \) is defined so that:

\[
\psi'(x, y) := \phi(x) \cdot \psi(y), \quad \text{for each } x, y \text{ of } X \cup Y. \tag{3.3}
\]

The commutative law applies to multiplication of potentials, i.e., \( \phi_1 \phi_2 = \phi_2 \phi_1 \), as does the associative law, i.e., \( (\phi_1 \phi_2)\phi_3 = \phi_1 (\phi_2 \phi_3) \). These laws also apply to addition of potentials, i.e., \( \phi_1 + \phi_2 = \phi_2 + \phi_1 \) and \( (\phi_1 + \phi_2) + \phi_3 = \phi_1 + (\phi_2 + \phi_3) \). The distributive law applies to addition of potentials as well, i.e., \( \phi \cdot (\psi_1 + \psi_2) = \phi \cdot \psi_1 + \phi \cdot \psi_2 \).

The division of two potentials is also used during the evaluation of an ID, and is similarly defined element-wise. The result of dividing a utility potential \( \psi(X) \) by a probability potential \( \phi(Y) \) is a utility potential \( \psi'(X, Y) \) defined as follows:

\[
\psi'(x, y) := \frac{\psi(x)}{\phi(y)}, \quad \text{for each } x, y \text{ of } X \cup Y. \tag{3.4}
\]
When both the numerator and denominator are zero, the convention \( \frac{0}{0} = 0 \) is adopted.

Two marginalization operators are used in the VE algorithm for IDs: sum-marginalization is used to eliminate chance variables, and max-marginalization is used to eliminate decision variables.

Given a probability potential \( \phi \), the elimination of a variable \( Y \) by sum-marginalization results in a new probability potential \( \phi' \) over the variables \( \text{dom}(\phi) \setminus Y \), defined as \( \phi' = \sum_Y \phi \). For example, given a potential \( \phi(Y, X) \), elimination of the variable \( Y \) by sum-marginalization creates a new potential \( \phi'(X) \) defined as follows:

\[
\phi'(x) := \sum_{y \in \text{sp}(Y)} \phi(y, x), \text{ for each } x \text{ of } X.
\] (3.5)

Similarly, the sum-marginalization \( \sum_Y \psi \) of a utility potential \( \psi(Y, X) \) is a utility potential over \( X \) such that

\[
\psi'(x) := \sum_{y \in \text{sp}(Y)} \psi(y, x),
\] (3.6)
for each \( x \in \text{sp}(X) \). The commutative law applies to sum-marginalization, i.e., \( \sum_Y \sum_Z \psi = \sum_Z \sum_Y \psi \). The distributive law also applies. That is, if \( Z \notin \text{dom}(\psi_1) \), then \( \sum_Z \varphi_1 \varphi_2 = \varphi_1 \sum_Z \varphi_2 \).

The max-marginalization, \( \max_Y \psi \), of a utility potential \( \psi \) is a new utility potential over \( \text{dom}(\psi) \setminus Y \). For example, given a potential \( \psi(Y, X) \), the elimination of variable \( Y \) by max-marginalization creates a new potential \( \psi'(X) \) defined as follows:

\[
\psi'(x) := \max_{y \in \text{sp}(Y)} \psi(y, x), \text{ for each } x \text{ of } X.
\] (3.7)

The max-marginalization of a probability potential is defined similarly. The distributive law applies to max-marginalization. That is, if \( Z \notin \text{dom}(\psi_1) \), then \( \max_Z \varphi_1 \varphi_2 = \)
the commutative property does not apply because the order of decision variables cannot be changed.

### 3.1.2 Pseudocode of Algorithm

For a given ID, let \( \{X_1, \ldots, X_p\} = U_C \cup U_D \) denote the set of all variables of the ID. The probability and utility functions of an ID can themselves be viewed as potentials. For each chance variable \( X_i \in U_C \), the conditional probability distribution \( P(X_i|pa(X_i)) \) is equivalent to a probability potential \( \phi : sp(\{X_i\} \cup pa(X_i)) \to [0, 1] \). For each value node \( V_j \), the corresponding utility function \( U_j \) is equivalent to a utility potential that is denoted \( \psi : sp(pa(V_j)) \to \mathbb{R} \).

When the conditional probability functions and utility functions of an ID are viewed as potentials, Equation (2.13) can be reformulated as follows,

\[
MEU = \max_{I_0} \left( \max_{D_1} \ldots \max_{D_n} \left( \prod_{I_n} \phi \sum_{\Psi} \right) \right)
\]

(3.8)

where \( \Phi \) denotes the set of probability potentials, \( \Psi \) denotes the set of utility potentials, and the expression \( \prod \Phi \sum \Psi \) is the product of all probability potentials multiplied by the sum of all utility potentials.

Algorithm 4 shows pseudocode for the traditional VE algorithm. The algorithm solves an ID by successively eliminating the variables in (3.8). Below, the key steps of the algorithm are summarized.

**Identify relevant potentials** After a variable \( X_i \) is selected for elimination, the first step is to identify the relevant potentials, which is accomplished in lines 7 and 8 of the pseu-
**Algorithm 4: Traditional variable elimination**

```plaintext
/* Initialize sets of probability and reward potentials */
Φ ← \{ P(C_i|pa(C_i)) | C_i ∈ C \};
Ψ ← \{ U(pa(V_j)|V_j ∈ V) \};

while there is a variable not yet eliminated do
  select a variable X_i eligible to be eliminated;
  /* get relevant probability and reward potentials */
  Φ_{X_i} ← \{ φ ∈ Φ | X_i ∈ dom(φ) \};
  Ψ_{X_i} ← \{ ψ ∈ Ψ | X_i ∈ dom(ψ) \};
  /* combine potentials */
  Q_φ^{X_i} ← Φ_{X_i};
  Q_ψ^{X_i} ← Φ_{X_i} ( Ψ_{X_i} )
  if X_i is a chance variable then
      /* eliminate X_i by sum marginalization */
      ϕ^{X_i}_X ← Σ_φ \{ ϕ \};
      ψ^{X_i}_X ← Σ_ψ \{ ψ \};
  else if X_i is a decision variable then
      /* eliminate X_i by max marginalization */
      ϕ^{X_i}_X ← max_ϕ \{ ϕ \};
      ψ^{X_i}_X ← max_ψ \{ ψ \};
      δ^X_i ← arg max_ψ \{ ψ \} /* save strategy */
  end
  /* update sets of potentials */
  Φ ← (Φ \ Φ_{X_i}) \cup {ϕ^{X_i}_X};
  Ψ ← (Ψ \ Ψ_{X_i}) \cup {ψ^{X_i}_X};
end
Δ^* = (δ^*_1, . . . , δ^*_n) /* optimal strategy */
```

docode. A potential is relevant if the selected variable X_i is in its domain. This step partitions the set Φ of probability potentials into two sets: the set of relevant probability potentials, denoted Φ_{X_i}, and the remaining probability potentials, denoted Φ^* = Φ \ Φ_{X_i}.

It also partitions the set Ψ of utility potentials into two sets: the set of relevant utility potentials, denoted Ψ_{X_i}, and the remaining utility potentials, denoted Ψ^* = Ψ \ Ψ_{X_i}.
Eliminate chance variable by sum-marginalization  If the variable $X_i$ selected for elimination is a chance variable, it is eliminated by sum-marginalization. For convenience, let $\phi'_X = \sum_X \prod \Phi_{x_i}$, and let $\psi'_X = \sum_X \prod \Phi_{x_i}(\sum \Psi_{x_i})$, as computed in lines 14 and 15 of the pseudocode, and note that these new potentials appear in the following derivation, which uses the distributive law:

$$
\sum_X \prod \phi \left( \sum \psi \right) = \prod \phi^* \sum_X \left( \prod \phi_{x_i} \left( \sum \phi^* + \sum \psi_{x_i} \right) \right)
\leq \prod \phi^* \sum_X \phi_{x_i} \left( \sum \phi^* + \sum \psi_{x_i} \right)
= \prod \phi^* \phi'_{x_i} \left( \sum \psi^* + \frac{\psi'_{x_i}}{\phi'_{x_i}} \right). \tag{3.9}
$$

This derivation shows that the result of eliminating a chance variable $X_i$ by sum-marginalization is that $\Phi_{x_i}$ is removed from the set of probability potentials, and replaced by the new probability potential $\phi'_X$, and $\Psi_{x_i}$ is removed from the set of utility potentials, and replaced by the new utility potential $\frac{\psi'_{x_i}}{\phi'_{x_i}}$. This computation is performed in stages by Algorithm 4, in lines 10 and 11, 14 and 15, and 22 and 23.

Eliminate decision variable by max-marginalization  If the variable $X_i$ selected for elimination is a decision variable, it is eliminated by max-marginalization. In this case, let $\phi'_X = \max_X \prod \Phi_{x_i}$, and let $\psi'_X = \max_X \prod \Phi_{x_i}(\sum \Psi_{x_i})$, as computed in lines 18 and
19, and note that these new potentials appear in the following derivation, which uses the distributive law:

\[
\max_{X_i} \prod \Phi \left( \sum \Psi \right) = \prod \Phi^* \max_{X_i} \left( \prod \Phi_{X_i} \left( \sum \Psi^* + \sum \Psi_{X_i} \right) \right) \\
= \prod \Phi^* \left( \max_{X_i} \prod \Phi_{X_i} \left( \sum \Psi^* \right) + \max_{X_i} \prod \Phi_{X_i} \left( \sum \Psi_{X_i} \right) \right) \\
= \prod \Phi^* \left( \sum \Psi^* + \sum \Psi_{X_i} \right) \\
= \prod \Phi^* \phi_{X_i} \left( \sum \Psi^* + \frac{\psi_{X_i}}{\phi_{X_i}} \right) 
\]

(3.10)

This derivation shows that the result of eliminating a decision variable \( X_i \) by max-marginalization is that \( \Phi_{X_i} \) is removed from the set of probability potentials, and replaced by the new probability potential \( \phi'_{X_i} \), and \( \Psi_{X_i} \) is removed from the set of utility potentials, and replaced by the new utility potential \( \psi_{X_i} / \phi_{X_i} \). This computation is performed in stages by Algorithm 4, in lines 10 and 11, 18 and 19, and 22 and 23.

Note that when eliminating a decision variable \( X_i \) by max-marginalization from the probability potential \( \phi_{X_i} = \prod \Phi_{X_i} \), there is often not any probability potential containing \( X_i \), or, if any, it is a vacuous potential. That is because each decision is d-separated from its predecessors, and any successor has already been removed (the removal order of the disjoint subsets of variables must respect the temporal constraints). As a result, the efficiency of the algorithm could be improved by not computing the maximum for the probability potential, and simply setting the variable \( X_i \) equal to the same value \( x_i \) that maximizes the utility potential.

The last lines of Equation (3.9) and Equation (3.10) show that eliminating a chance variable by sum-marginalization, or eliminating a decision variable by max-marginalization,
does not change the maximum expected utility (MEU) of the ID. Therefore, when the last variable is eliminated, the VE algorithm returns a potential with no arguments (i.e., a constant) that is the value of Equation (3.8), that is, it is the MEU.

**Strategy representation** For each decision variable $D_i$ that is eliminated, a decision function $\delta_{D_i}$ is created in line 20 of the algorithm that is defined on the set of relevant variables, and gives the optimal decision for each configuration of values of the relevant variables. That is, each decision rule $\delta_i$ is a mapping, $\delta_i : sp(pa(D_i)) \rightarrow sp(D_i)$. Upon termination of the VE algorithm, an optimal strategy $\Delta = (\delta_1, \ldots, \delta_n)$, has been computed.

### 3.1.3 Elimination Ordering Heuristics

Unlike the VE algorithm for Bayesian networks, there are restrictions on the order in which variables can be eliminated. In particular, variables must be eliminated in reverse order of the partial ordering imposed by the information constraints shown in Equation (2.1), which is called a *strong elimination ordering* [25], and is summarized below:

$$I_n \rightarrow D_n \rightarrow I_{n-1} \rightarrow \ldots \rightarrow D_1 \rightarrow I_0.$$  \hspace{1cm} (3.11)

That means that VE first sum-marginalizes $I_n$, then max-marginalizes $D_n$, then sum-marginalizes $I_{n-1}$, etc.

Within each set of chance variables $I_i$, however, any ordering of the chance variables is allowed because sum-marginalization is commutative, and a good ordering can improve the efficiency of the algorithm. As is the case for VE for Bayesian networks, the problem of finding an optimal ordering is an NP-hard problem. In practice, an effective approach is to
use a greedy heuristic that selects the next variable to eliminate in an order that minimizes the size of the largest potential created.

3.2 Generalized Variable Elimination Algorithm

With this background, we are ready to introduce a more general approach to variable elimination for IDs that integrates the traditional variable elimination algorithm for IDs with the incremental pruning algorithm for POMDPs [10]. We call this new algorithm Generalized Variable Elimination. Although the class of problems it solves is the same as the class of problems solved by traditional VE, it can solve these problems more efficiently by using dynamic programming techniques originally developed for solving POMDPs.

The key idea of this generalization is that the traditional constraints on the order in which variables are eliminated by the VE algorithm can be relaxed by using dynamic programming techniques for POMDPs that allow reasoning about hidden state variables. We explain the details of this generalization below.

3.2.1 New Strategy Representation for Influence Diagrams

We adopted the belief states representation of strategy from the POMDP literature and made a couple of changes on it to let it fit into IDs’ framework. Since a strategy is conditioned of both previous decisions and observations, it is useful to distinguish between chance variables that are observable (the sets $I_0, I_1, \ldots, I_{n-1}$) and chance variables that are not observable (the set $I_n$). To indicate this distinction, we call chance variables that are observable observation variables and chance variables that are not observable hidden variables.
POMDP approach defines strategy stage by stage since it solve the problem stage by stage, and for each stage of the problem, there are hidden variables, decision variables, and observed variables. Therefore, the definition of the strategy in POMDPs shown in Equation (2.16) includes decision and observed variables together. However, IDs’ approaches instead treat problems stage by stage, they treat problems variable by variable, and they removes one variable each step.

Let \( T \) denote the total number of decision and observation variables, and let \( X_T, X_{T-1}, \ldots, X_1 \) denote a valid ordering of these variables, that is, an ordering consistent with the information constraints of the problem. In this ordering, the subscript \( t \) of \( X_t \) indicates the number of stages until the end of the problem. Thus any instantiation of the sequence \( X_T, X_{T-1}, \ldots, X_1 \) corresponds to a valid history, and the set of all instantiations is the set of all possible histories.

We can recursively define the set \( S_t \) of \( t \)-stage strategies, where a \( t \)-stage strategy \( s_t \in S_t \) specifies a strategy for the last \( t \) stages of the problem. The base case is simply:

\[
S_0 = \{ \text{stop} \},
\]

where \( \text{stop} \) is an artificial action with no effect that simply terminates the problem; by default, it has a utility of zero.

The recursive part of the definition has two cases. If \( X_t \) is a decision variable, then:

\[
S_t = \{ (x_t, \text{succ}_t) | x_t \in X_t, \text{succ}_t : \{ \text{null} \} \rightarrow S_{t-1} \},
\]

where \( x_t \) is the choice of an action, which is appended to a \((t-1)\)-stage strategy. In this case, the number of possible strategies in \( S_t \) is \(|X_t||S_{t-1}|\).
If $X_t$ is an observation variable, then

$$S_t = \{\langle \textsf{null}, \text{succ}_t \rangle \mid \text{succ}_t : X_t \rightarrow S_{t-1}\}.$$  \hfill (3.14)

For each $t$-stage strategy $s_t \in S_t$, a distinct function $\text{succ}_t$ maps each observation to a successor $(t-1)$-stage strategy. Note that in the case of a decision variable, we let $\textsf{null}$ denote a null observation that provides no information. In the case of an observation variable, we let $\textsf{null}$ denote a null action that has no effect. When $X_t$ is an observation variable, the number of possible strategies in $S_t$ is $|S_{t-1}|^{|X_t|}$.

**Strategy and Strategy Variable**

We use $S_i$ to denote the strategy variable for stage $i$. The domain of $S_i$, denote as $S_i$, is a set of all strategies that are included in stage $i$. For our generalized variable elimination algorithm, a set of strategies $S_i$ for stage $i$ is recursively defined by Equations (3.12), (3.13) and (3.14), where a strategy set $S_i$ is created from the selected variable $X_i$ and the previous strategy set $S_{i-1}$. We use $s_i$ to denote a single strategy for stage $i$ and $s_i \in S_i$.

We also define a mapping function: $\delta_i : h \rightarrow S_{i,h}$, which means function $\delta_i(h)$ returns a set of strategies $S_{i,h}$, where $S_{i,h}$ is the set of strategies for a given history instantiation $h$ for stage $i$. Notice that, $S_{i,h} \subseteq S_i$ and $\bigcup \{\delta_i(h), \forall h \in H\}$ is $S_i$.

Compared to other regular variables in the ID, the strategy variable is special in such ways. First, strategy variables are not a part of the original influence diagram, and they are artificial variables that we added to help the algorithm to evaluate the problem, and also to store the solution. Second, we create one strategy variable $S_i$ for each stage $i$, and use it to
indicate the set of strategies for that stage. After the influence diagram has been solved, all
the strategy variables together represent the solution for the problem.

By introducing the concept of the strategy variable, we can represent strategies explicitly in the GVE algorithm. At the beginning of the evaluation, there is only one strategy in the strategy variable, and that strategy is the stop action that defined in Equation (3.12). After that, the strategies in the strategy variable will change as the algorithm running, and we will explain the details of how to change the strategies in later sections.

**Strategy Variable and Utility Potential**

Intuitively, the relationship between a single strategy and a utility potential is straightforward, that is, for each strategy, there is a utility potential associated with it. In other words, they are one to one corresponding. Therefore, the strategy variable can be included as an additional variable in the domain of the utility potential. Let \( U \) denotes the set of variables that are hidden chance variables that in the domain of the utility potential \( \psi \), and let \( H \) denotes the set of variables that are decision and observation variables that in the domain of the utility potential \( \psi \). For each strategy \( s_i \in S_i \), each instantiation \( u \in U \) and each instantiation \( h \in H \), there is a scalar value \( v = \psi(s_i, u, h) \).

**Piece-wise linear and convex value function representation in utility potential**

By adopt the similar concepts of piece-wise linear and convex value function representation in POMDPs. A utility potential \( \psi(s_i, h, U) \) can be considered as a single real-valued \(|X|\)-dimensional vectors, denotes as \( v_i \), where \( X \) is the Cartesian product of all the hidden chance variables in \( U \) for history instantiation \( h \). And \( \psi(S_i, h, U) \) can be considered as a set \( S_{i,h} \) of real-valued \(|X|\)-dimensional vectors, denotes as \( V_{i,h} \), for history instantiation \( h \).
Therefore, $\psi(S_i, H, U)$ that also can be written as $\psi_i$ (the utility potential that created by stage $i$) can be considered as a set $S_i$ of real-valued $|X|$-dimensional vectors, denotes as $\mathcal{V}_i$ for all histories.

Let $\Xi(h)$ represents the value function for a history instantiation $h$. Notice that, once we adopt the piece-wise linear and convex value function representation, $\Xi(h)$ not only can represent a value function for a particular prior probability distribution over $U$ for a history instantiation $h$, but also can represents the value function for any prior probability distribution over $U$ for a history instantiation $h$. Moreover, for any belief state $b$ over instantiations of the hidden state variables $U$ (that is, for any probability distribution over instantiations $u \in U$), and for one instantiation $h \in H$ of the related decision and observation variables, the value function of one strategy $s_i$ is $\sum_{u \in U} b(u)\psi(s_i, h, u)$. Finally, for any belief state $b$ over instantiations $u \in U$ of the hidden state variables, and for one history instantiation $h \in H$, the optimal value function over a set of strategies is:

$$\Xi(b, h) = \max_{s_i \in S_{i,h}} \sum_{u \in U} b(u)\psi(s_i, h, u).$$

(3.15)

where $S_{i,h}$ is a set of strategies by given $h$ at stage $i$.

**Eliminate Dominated Strategies**

Eliminate dominated strategies, called *prune* is a subroutine that is called by GVE, and it does the same process for both decision and chance variables. The algorithm executes prune for each history instantiation $h$ individually. There are two cases will happen when eliminating dominated strategies. In the first, $U$ is empty, that is, the utility potential does
not depend on any of the hidden chance variable. This case is relatively easy: a strategy \( s_t^i \in S_i \) with corresponding value vector \( v_t^i \) is dominated if

\[
v_t^i(h) \leq \max_{v_t^k \in V_i \setminus \{v_t^i\}} v_t^k(h), \forall h \in H,
\]

(3.16)

In this case, \( v_t^i \) is a scalar value. Therefore, Equation (3.16) finds a single strategy that has the maximum scalar value.

In the second case, \( U \) is non-empty, that is, the utility potential depends on hidden chance variables. This case requires reasoning about hidden state in a way that generalizes the pruning step of the incremental pruning algorithm for POMDPs. Given a set \( S_i \) of \( i \)-stage strategies with corresponding value vectors \( V_i \), a strategy \( s_t^i \in S_i \) is dominated by the other strategies \( S_i \setminus \{s_t^i\} \) if for all belief states \( b \):

\[
\sum_{u \in U} b(u)v_t^i(u, h) \leq \max_{v_t^k \in V_i \setminus \{v_t^i\}} \sum_{u \in U} b(u)v_t^k(u, h), \forall h \in H,
\]

(3.17)

Because this test considers all possible belief states, that is, all probability distributions over the hidden states, it requires use of linear programming. The algorithm for prune dominated vectors is the Lark’s algorithm that we discussed in Chapter 2. In the worst case, it solves a linear program for each strategy \( s_t^i \in S_i \).

### 3.2.2 Main Algorithm

The frame of generalized variable elimination algorithm is similar to the traditional variable elimination algorithm, the pseudocode of GVE shows in Algorithm 5. First, it takes CPTs and utility tables as inputs, transforms them into probability and utility potentials, shows in line 7-8. Then GVE removes one variable each step, shown in line 9-38.
Algorithm 5: Generalized Variable elimination

1 Function GVE (A legitimate elimination ordering: X = {X₁, X₂, …, X_p})

/* create a strategy variable S₀ and a strategy function δ₀ that only have one strategy stop */
δ₀ <- {stop}

/* create a utility potential function ψ₀(stop), and set it the values to 0 */
ψ₀(stop) = 0

/* Initialize sets of probability and utility potentials */
Φ ← {P(Cᵢ|pa(Cᵢ))|Cᵢ ∈ C}
Ψ ← {U(pa(Vᵢ)|Vᵢ ∈ V}

for i ← 0 upto p do // eliminate variable Xᵢ

// Get potentials that depend on Xᵢ
Φₓᵢ ← {φ ∈ Φ|Xᵢ ∈ dom(φ)}
Ψₓᵢ ← {ψ ∈ Ψ|Xᵢ ∈ dom(ψ)}

Y ← dom(Φₓᵢ) ∪ dom(Ψₓᵢ) \ Xᵢ /* Relevant variables for Xᵢ */
H ← Decision variables and Observed chance variables ∈ Y
U ← Hidden chance variables ∈ Y

/* if the variable does not involve utility potentials, then remove it won’t have any affect on strategies at this stage */
if ψₓᵢ = ∅ then

    /* simple update the strategy stages index, nothing change. */
    δᵢ ← δᵢ₋₁
    and ψₓᵢ = ψₓᵢ₋₁

    /* update the probability potential only */
    if Xᵢ is a chance variable then
        φ’ₓᵢ ← Σₓᵢ, ∏Φₓᵢ,
    if Xᵢ is a decision variable then
        φ’ₓᵢ ← ∏Φₓᵢ=xᵢ

else

    /* add utility potential from last stage (include strategy variable) in to the set of utility potential */
    if ψₓᵢ₋₁ ∉ Ψₓᵢ then
        Ψₓᵢ ← Ψₓᵢ ∪ ψₓᵢ₋₁
    if Xᵢ is a hidden chance variable then
        {φ’ₓᵢ, ψ’ₓᵢ} ← GSM(Φₓᵢ, Ψₓᵢ, Xᵢ, H, U, δᵢ₋₁)
    else if Xᵢ is a decision variable then
        {φ’ₓᵢ, ψ’ₓᵢ} ← UM(Φₓᵢ, Ψₓᵢ, Xᵢ, H, U, δᵢ₋₁)
    else if Xᵢ is a observed chance variable then
        {φ’ₓᵢ, ψ’ₓᵢ} ← CSM(Φₓᵢ, Ψₓᵢ, Xᵢ, H, U, δᵢ₋₁)

/* update sets of potentials */
Φ ← (Φ \ Φₓᵢ) ∪ {φ’ₓᵢ}
Ψ ← (Ψ \ Ψₓᵢ) ∪ {ψ’ₓᵢ}

end

return {S_p, ψ’ₓ_p}
When a variable has been eliminated, GVE creates a new probability potential $\phi'$ and a new utility potential $\psi'$, and updates the probability potentials and utility potentials sets using $\phi'$ and $\psi'$, shows in line 36-37. A key difference compared to traditional VE is that GVE also creates a new strategy variable for each stage using the strategy variable from the last stage, and notice that, the utility potentials in GVE depend on strategy variables.

**Initialization Step**

At the beginning of the evaluation, GVE creates the strategy variable $S_0$, and a strategy function $\delta_0$. There is only one strategy $stop$ in $S_0$, and there is no history involved, shows in line 3. GVE also creates a utility potential, denote as $\psi(S_0)$. Since $S_0$ only have one strategy $stop$, $\psi(S_0) = \psi(stop)$, and we initially set $\psi(stop) = 0$, shows in line 5. If we add $\psi(stop)$ to another utility potential $\psi_{X_1}$, and get a new utility potential $\psi_{combine}$. The value of $\psi_{combine}$ won’t chance, but the domain of the $\psi_{combine}$ is changed, it become $\{S_0 \cup dom(\psi_{X_1})\}$. As a result of that, GVE successfully creates a utility potential that depends on strategy variable without changing the MEU of the ID.

**New Variable Elimination Ordering**

One significant elimination ordering constraint relaxation is that GVE does not request strong elimination order any more. In other words, it does not have to eliminate all the hidden chance variables before removing the last decision variable. However, GVE still following the reverse temporal ordering when removing decision and observation variables.

Besides that, there is also another constraint on elimination ordering that is, when eliminating a decision variable, we do not allow any hidden chance variables that is a successor
of this decision variable. If the decision variable has a successor that is a hidden chance variable, we need to remove the hidden chance variable first. This constraint combined with the reverse temporal ordering constraint together guaranteed that there is no probability potential depends on strategies. In other words, the probability potential is independent of strategies. This fact simplified the algorithm a lot, and allow GVE handle probability potential exactly the same as traditional VE.

The elimination ordering plays a very important role in GVE since it directly affects the time complexity of the algorithm. We show for some IDs, this relaxation of elimination ordering leads ten orders of magnitude speedup in term of the algorithm running time. However, determine a good elimination order is not easy, we will discuss the heuristic of how to decide an elimination order in the later section.

After selecting which variable is going to remove, the algorithm identifies the relevant potentials and the set of variables $Y$ that belongs to the domain of those potentials, shows in line 11-13. Then, the algorithm distinguishes the set of history variables (decision and observation variables), denote as $H$ and the set of hidden chance variables, denote as $U$, shows in line 14, 15.

GVE create a strategy variable $S_i$ and a strategy function $\delta_i$ for each stage $i$. However, for some stages, the variable that is going to eliminate only involves probability potentials. In that case, remove those variables won’t affect either the strategies nor the utility potential that depend on the strategies. Therefore, we simply update the index of strategy variable, mapping function and the utility potential that involve the strategy variable, but do not
chance the value of them. Then we update the probability potentials just like the traditional
VE, shows in line 17-24.

Finally, if the variable that is going to remove involve utility potential. First, check
if the relevant utility potentials already include the utility potential \( \psi_{X_{i-1}} \) that contains
strategy variable from last stage, if not, add \( \psi_{X_{i-1}} \) into \( \Psi_{X_i} \), shows in line 27, 28. This
step make sure that strategy variables will be included every time GVE modifies the utility
potentials. Then, GVE distinguish between hidden chance variable and observed chance
variable, and depend on which kind of variable this current stage is going to remove, the
algorithm call one of the three functions GSM, UM or CSM, shows in line 29-34.

### 3.2.3 Generalized Sum-Marginalization

The algorithm executes function Generalized Sum-Marginalization(GSM) when elimi-
nating hidden chance variables, the pseudocode for GSM shows in Algorithm 6. The GSM
step for GVE is similar to the sum-marginalization step in the traditional VE that it creates
a new probability potential and a new utility potential that does not depend on the currently
hidden chance variable.

GSM first combines the set of relevant potentials together, and create a new single
probability potential \( \phi_{X_i} \) and a new single utility potential \( \psi_{X_i} \), shows in line 2-4. The new
probability potential \( \phi'_{X_i} \) that does not depend on \( X_i \) is created by summation over each
instantiation \( x_i \) of \( X_i \), shows in line 8.

After that, GSM does two jobs, first, creates a new set of strategies, and second, creates
a new utility potential that depends on the new sets of strategies but does not depend on
Algorithm 6: Generalized Sum Marginalization

1 Function GSM (Φ_Xi, Ψ_Xi, X_i, H, U, δ_{i-1})
2 /* combine potentials */
3 \( \phi_{X_i} \leftarrow \prod \Phi_{X_i} \)
4 \( \psi_{X_i} \leftarrow \prod \Phi_{X_i} \left( \sum \Psi_{X_i} \right) \)
5 foreach instantiation \( h \) of \( H \) do
6 foreach instantiation \( u \) of \( U \) do
7 /* new probability potentials, does not depend on strategy variable */
8 \( \phi'_{X_i}(h, u) = \sum_{x_i \in X_i} \phi_{X_i}(h, x_i, u) \)
9 end
10 \( \delta_i(h) = \emptyset \)
11 foreach strategy \( s_{i-1} \in \delta_{i-1}(h) \) do
12 \( s_i \leftarrow s_{i-1} \) /* create a new strategy \( s_i \) for stage \( i \) (not change)*/
13 \( \delta_i(h) \cup s_i \) /* adding new strategy to strategy function */
14 /* update the utility potential for each strategy */
15 foreach instantiation \( u \) of \( U \) do
16 \( \psi'_{X_i}(s_i, h, u) = \frac{\sum_{x_i \in X_i} \psi_{X_i}(x_i, s_{i-1}, h, u)}{\phi'_{X_i}(h, u)} \)
17 end
18 end
19 \( \delta_i(h) = \text{prune}(\delta_i(h)) \)
20 end
21 return \( \{ \phi'_{X_i}, \psi'_{X_i} \} \)
Since removing hidden chance variables do not affect strategies, GSM simply assigns the new strategy using the strategy from the last stage and add it to the new strategies set, shows in line 12-13.

For each new strategy \( s_i \), each history variables instantiation \( h \), and each hidden variables instantiation \( u \), the new scalar value is computed by:

\[
\psi_{X_i}(s_i, h, u) = \sum_{x \in X_i} \frac{\psi_{X_i}(x_i, s_{i-1}, h, u)}{\phi_{X_i}(h, u)}
\]

where \( s_{i-1} \) is the strategy from last stage, \( x_i \) is the a single state of \( X_i \) and \( X_i \) is the hidden chance variable that needs to eliminate, shows in line 16. From Equation (3.18), we know that it is doing summation over \( X_i \) for each strategy \( s_{i-1} \) to create a new utility potential that does not depend on \( X_i \) and \( H \) and \( U \) in the new utility potential are remained the same.

The last step is to eliminate dominated strategies since the value function corresponding to each strategy is changed after eliminating the hidden chance variable. As we mentioned before, \( \psi_{X_i}(S_i, h, U) \) can be considered as a set \( \delta_i(h) \) of real-valued \( |X| \)-dimensional vectors, denotes as \( V_{i,h} \), for history instantiation \( h \). Therefore, we can apply Equation (3.16) or Equation (3.17) to prune dominated strategies for each history instantiation \( h \), shows in line 19, and prune function executes Lark’s algorithm.

### 3.2.4 Union-Marginalization

The algorithm executes function Union-Marginalization (UM) when eliminating decision variables, the pseudocode for function UM shows in Algorithm 7. Union-Marginalization creates a new set of strategies for each history base on previous set of
Algorithm 7: Union Marginalization

1. **Function** $UM(\Phi_{X_i}, \Psi_{X_i}, X_i, H, U, \delta_{i-1})$
2. /* combine potentials */
3. $\phi_{X_i} \leftarrow \prod \Phi_{X_i}$
4. $\psi_{X_i} \leftarrow \prod \Phi_{X_i}(\sum \Psi_{X_i})$
5. **foreach** instantiation $h$ of $H$ do
6. **foreach** instantiation $u$ of $U$ do
7. /* $x_i \leftarrow$ arbitrary value of $X_i$, since probability does vary for different action. */
8. $\phi'_{X_i}(h, u) = \phi_{X_i}(x_i, h, u)$
9. end
10. $\delta_i(h) = \emptyset$
11. **foreach** instantiation $x_i$ of $X_i$ do
12. **foreach** strategy $s_{i-1} \in \delta_{i+1}(h, x_i)$ do
13. $s_i \leftarrow (x_i, s_{i-1})$ /* create new strategy */
14. $\delta_i(h) \cup s_i$ /* adding new strategy to strategy function */
15. /* update the utility potential */
16. **foreach** instantiation $u$ of $U$ do
17. $\psi_{X_i}^{\text{new}}(s_i, h, u) = \frac{\psi(x_i, s_{i-1}, h, u)}{\phi'_{X_i}(h, u)}$
18. end
19. end
20. $\delta_i(h) = \text{prune}(\delta_i(h))$
21. end
22. **return** $\{ \phi'_{X_i}, \psi'_{X_i} \}$
strategies and the actions that are taken by the current decision variable for that history. It also creates a new utility potential that corresponds to the new set of strategies.

First, UM combines the set of relevant potentials together, and create a new single probability potential \( \phi_{X_i} \) and a new single utility potential \( \psi_{X_i} \), shows in line 2-4. The new probability potential \( \phi'_{X_i} \) does not vary with \( X_i \) since all variables coming after \( X_i \) in the temporal ordering have been eliminated. Therefore, the value of \( \phi'_{X_i}(h, u) \) is created by assign the value from an arbitrary instantiation \( x_i \) of \( X_i \) in \( \phi_{X_i}(X_i, h, u) \), shows in line 8.

Instead of picking the action that leads to the maximum expected utility (scalar value) then threw away other actions like the traditional max-marginalization does for decision variable in VE, as the name suggested, UM first creates a set of new strategies by combining the current action with all possible previous strategies, called action-strategy set, then union all the new strategies in different action-strategy sets. Shows in line 13-14 in Algorithm 7. Those strategies that created by UM include all the possible strategies for the future from this point.

It is interesting to note that UM step does not explicitly eliminate a decision variable \( X_i \) by maximization. But it implicitly does so, for both decision variables and observation variables, by maximizing over the strategy set \( S_i \). At the end of the algorithm, after all other variables have been eliminated, the strategy variable becomes the last variable eliminated, and then we select the optimal strategy with the maximum value.
Now, let us see how to create the new utility potential. For each new strategy $s_i$, each history variables instantiation $h$, and each hidden variables instantiation $u$, the new scalar value is computed by:

$$
\psi_{X_i}^{new}(s_i, h, u) = \frac{\psi(x_i, s_{i-1}, h, u)}{\phi_{X_i}(h, u)}
$$

where $x_i$ is the current action, and $s_{i-1}$ is the previous strategy that under history instantiation $(h, x_i)$, shows in line 17.

The last step for UM is to eliminate dominated strategies since the set of strategies is changed after eliminating the decision variable. As we mentioned before, $\psi'_X(S_i, h, U)$ can be considered as a set $\delta_i(h)$ of real-valued $|X|$-dimensional vectors, denotes as $V_{i,h}$, for history instantiation $h$. Therefore, we can apply Equation (3.16) or Equation (3.17) to prune dominated strategies for each history instantiation $h$, shows in line 21, and prune function executes Lark’s algorithm.

### 3.2.5 Crosssum-Marginalization

The algorithm executes function Cross-Sum-Marginalization (CSM) when eliminating observed chance variables, the pseudocode for CSM shows in Algorithm 8. As its name suggested, cross-sum-marginalization is a combination of sum-marginalization operation in traditional VE algorithm and cross-sum operation in incremental pruning algorithm. Cross-Sum-Marginalization creates a set of new strategies for each history instantiation $h$ base on the previous set of strategies for history instantiations $(h, X_i)$. It also creates a new utility potential that corresponds to each new strategy.
Algorithm 8: Cross Sum Marginalization

Function CSM (Φᵢ, Ψᵢ, Xᵢ, H, U, δᵢ₊₁)

/* combine potentials */
φᵢ ← ∏ Φᵢ
ψᵢ ← ∏ Φᵢ(∑ Ψᵢ)

foreach instantiation h of bf H do
    δᵢ(h) ← ∅
    foreach instantiation xᵢ of Xᵢ do
        foreach instantiation u of U do
            φᵢ'(h, u) = φᵢ(h, u) + φᵢ(xᵢ, h, u) /* update probability potentials */
        end
    end

/* prepare for cross sum, clear up space to store new strategy */
δₚₑᵗₚₑₜ(h) ← δᵢ(h) and δᵢ(h) ← ∅
foreach strategy sᵢ ∈ δₚₑᵗₚₑₜ(h) do
    foreach strategy sᵢ₋₁ ∈ δᵢ₋₁(h, xᵢ) do
        sᵢ = (sᵢ ∪ sᵢ₋₁) /* create new strategy */
        δᵢ(h) ∪ sᵢ /* adding new strategy to strategy function */
        /* update the utility potential */
        foreach instantiation u of U do
            ψᵢ'(sᵢ, h, u) = ψᵢ'(sᵢ, h, u) + ψᵢ(xᵢ, sᵢ₋₁, h, u)
        end
    end
    δᵢ(h) = prune(δᵢ(h))
end
/* normalize utility potential */
foreach strategy sᵢ ∈ δᵢ(h) do
    foreach instantiation u of U do
        ψᵢ'(sᵢ, h, u) = ψᵢ'(sᵢ, h, u) / φᵢ(h, u)
    end
    δᵢ(h) = prune(δᵢ(h))
end
return { φᵢ', Ψᵢ' }
First, CSM combines the set of relevant potentials together, and create a new single probability potential $\phi_{X_i}$, and a new single utility potential $\psi_{X_i}$, shows in line 2-4. Since probability potentials does not depend on strategies, the calculation for creating new probability potentials is the same as the traditional VE. The for loop on $h$, $x$, $u$ and the equation in line 9 together computes the new probability potential.

The way our algorithm constructs new strategies for an observed chance variable has a significant difference compared to traditional VE. In traditional VE, an observed chance variable is considered a part of history, and it does not represent strategy itself. However, as we discussed before, in the GVE an observed chance variable has its strategies, defined in Equation (3.14). Those strategies that created by CSM include all the possible strategies for the future from this point.

Although the idea from Equation (3.14) of construct strategy is straightforward, the naive implementation of construct strategies is not accepted due to the cardinality of the set of strategies grows much more explosively when an observation chance variable is removed. Therefore, we adopt the cross sum step in the incremental pruning algorithm [10], and it can be performed more efficiently by interleaving generating and pruning of the new strategies. Let us see the detail of how CSM works.

The main idea of the cross-sum step is eliminating dominated strategies while constructing the strategies, and do it as early as possible. Let us introduce the concept of intermedia strategies $s_{temp}$ that is an uncompleted strategy, and there is an value function $v_{temp}$ corresponding to each $s_{temp}$. We first discuss how to construct the uncompleted strategy. When the loop on $X_i$ is not finished, the $s_i$ ,shows in line 15, is an uncompleted
strategy, and we copy it to $s_{temp}$, shows in line 12. When the loop on $X_i$ finished, the $s_i$ become complete strategy, Line 12, 15 and 16 put together shows the how to construct the new strategy based on the Equation (3.14). Notice that, we do not wait until the strategy has been constructed completely, then prune dominate strategies, we prune them as early as possible.

Now, let us see how to update the utility value. For each new strategy $s_i$, each history variables instantiation $h$, and each hidden variables instantiation $u$, the new scalar value is computed by:

$$\psi'_{X_i}(s_i, h, u) = \psi'_{X_i}(s_t, h, u) + \psi_{X_i}(x_i, s_{i-1}, h, u)$$

(3.20)

where $s_t$ is the intermedia strategy, $s_{i-1}$ is the strategy from last stage, $x_i$ is the a single state of $X_i$ and $X_i$ is the hidden chance variable that needs to eliminate, shows in line 19.

The last step is to eliminate dominated strategies since both the set of strategies and the value function corresponding to each strategy are changed after eliminating the obversed chance variable. As we mentioned before, $\psi'_{X_i}(S_t, h, U)$ can be considered as a set $\delta_i(h)$ of real-valued $|X|$-dimensional vectors, denotes as $V_{i,h}$, for history instantiation $h$. Therefore, we can apply Equation (3.16) or Equation (3.17) to prune dominated strategies for each history instantiation $h$, shows in line 19, and prune function executes Lark’s algorithm.

### 3.2.6 Algorithm Illustration

We illustrate Generalized variable elimination algorithm by solving the classic used car buyer problem [23, 48, 55] we discussed in Chapter 2. The most significant difference
between the traditional VE algorithm and our generalization is that the traditional algorithm eliminates all unobservable variables before any of the decision or observation variables.

For comparison, we are going to show 2 different elimination orderings for this problem. First, we show how our new algorithm can solve the used car buyer problem by eliminating all of the decision and observation variables before eliminating the unobservable variable C, which represents the condition of the car. Notice that, the first elimination ordering are only possible for our generalized VE. Second, we show how our new algorithm performance when applying traditional elimination ordering which is remove unobservable variable C first.

**First Elimination Ordering**

We using elimination ordering: \( B \rightarrow R_2 \rightarrow T_2 \rightarrow R_1 \rightarrow T_1 \rightarrow C \). This elimination ordering is only possible when using the generalized variable elimination algorithm. We initialized the MEU for this problem using Equation (3.8) and we get:

\[
MEU = \max_{t_1 \in T_1} \sum_{r_1 \in R_1} \max_{t_2 \in T_2} \sum_{r_2 \in R_2} \max_{b \in B} \sum_{c \in C} \left( P(c)P(r_1|c,t_1)P(r_2|c,t_1,r_1,t_2)U(c,t_1,t_2,b) \right).
\] (3.21)

**Initialization Step**

The prior probabilities for the variable \( C \) provide an initial belief state for the problem. In the POMDP approach we adopt, a problem can be solved for all belief states before considering the initial belief state. We create an artificial strategy variable \( S_0 \), and there is only one strategy \( s_0 \) in strategy set \( S_0 \) that is the strategy \( \text{stop} \). We also create a new utility potential \( \psi_0(S_0) \) and set its value to 0, denotes as \( \psi_0(s_0) = 0 \). As
we introduce before, we use \( \psi(c, t_1, t_2, b, s_0) \) to represent the utility potential that depend on strategy variable. For a given probability distribution \( b(c) \) (belief state), the value is:

\[
\max_{s_0 \in S_0} \sum_{c \in C, t_1, t_2, b} \psi(c, t_1, t_2, b, s_0).
\]

Therefore, we substitute \( U(c, t_1, t_2, b) \) with \( \psi(c, t_1, t_2, b) \), and get:

\[
MEU = \max_{t_1 \in T_1} \sum_{r_1 \in R_1} \max_{t_2 \in T_2} \sum_{r_2 \in R_2} \max_{b \in B} \sum_{s_0 \in S_0} \sum_{c \in C} \left( P(c) P(r_1 | c, t_1) P(r_2 | c, t_1, r_1) \psi(c, t_1, t_2, b, s_0) \right). \tag{3.22}
\]

based on:

\[
\psi(c, t_1, t_2, b, s_0) = U(c, t_1, t_2, b) + \psi_0(s_0) \tag{3.23}
\]

**Eliminating Variable \( B \)**

The first variable eliminated is the decision variable \( B \), and we apply union-marginalization, which leads to creation of the 1-stage strategy variable \( S_1 \). The only potential in Equation (3.22) that relevant to decision \( B \) is \( \psi(c, t_1, t_2, b, s_0) \).

After applying union-marginalization operation, we have a new utility potential

\[
\max_{s_1 \in S_1} \psi_B(c, t_1, t_2, s_1) = \max_{b \in B} \max_{s_1 \in S_1} \psi(c, t_1, t_2, b, s_0),
\]

where the strategy set \( \bar{S}_1 \) include all strategies that created by function UM. Then we eliminate the dominated strategies in set \( \bar{S}_1 \), and we get a minimum set of strategies \( S_1 \) that are not dominated by others (in this case, the strategies are the three possible actions are to buy the car with a guarantee, to buy the car without a guarantee, or not to buy the car at all, as shown in Figure 3.1).
Notice that, $\delta_1(t_1, t_2)$ may have more than one strategy and it may have a different set of strategies compared to $\delta_1(t'_1, t'_2)$. We update the MEU as follows:

$$MEU = \max_{t_1 \in T_1} \sum_{t_2 \in T_2} \max_{r_2 \in R_2} \sum_{s_1 \in S_1} \sum_{c \in C} \left( P(c) P(r_1|c, t_1) P(r_2|c, r_1, r_1, t_2) \psi_B(c, t_1, t_2, s_1) \right).$$

(3.24)

**Eliminating Variable $R_2$**

The next variable eliminated is the observation variable $R_2$ and the algorithm executes crosssum-marginalization operation, which leads to creation of strategy variable $S_2$ of 2-stage strategies, strategy set $\bar{S}_2$ initially has 27 strategies, and after pruning the dominated strategies, we get the minimum set $S_2$ that has 5 undominated strategies. We update the MEU as follow:

$$MEU = \max_{t_1 \in T_1} \sum_{r_1 \in R_1} \max_{t_2 \in T_2} \sum_{s_2 \in S_2} \sum_{c \in C} \left( P(c) P(r_1|c, t_1) \phi_{R_2}(c, t_1, r_1, t_2) \psi_{R_2}(c, t_1, t_2, s_2) \right)$$

(3.25)

based on the equivalence:

$$\max_{s_2 \in S_2} \psi_{R_2}(c, t_1, r_1, t_2, s_2) = \max_{s_1 \in S_1} \sum_{r_2 \in R_2} P(r_2|c, r_1, t_2) \psi_B(c, t_1, t_2, s_1)$$

(3.26)

Since the new probability potential

$$\phi_{R_2}(c, t_1, r_1, t_2) = \sum_{r_2 \in R_2} P(r_2|c, t_1, r_1, t_2)$$

(3.27)

**Eliminating Variable $T_2$**

The decision variable $T_2$ is eliminated next, leading to creation of a strategy variable $S_3$ of 3-stage strategies. The strategy set $\bar{S}_3$ initially has 10 strategies, and after pruning the
dominated strategies, we get the minimum set $S_3$ that has 3 undominated strategies. We update the MEU as follow:

$$MEU = \max_{t_1 \in T_1} \max_{r_1 \in R_1} \sum_{s_3 \in S_3} \sum_{c \in C} P(c) P(r_1|c,t_1) \phi_{T_2}(c,t_1,r_1) \psi_{T_2}(c,t_1,r_1,s_3)$$  \hspace{1cm} (3.28)$$

based on the equivalence

$$\max_{s_3 \in S_3} \psi_{T_2}(c,t_1,r_1,s_3) = \max_{t_2 \in T_2} \max_{s_2 \in S_2} \phi_{R_2}(c,t_1,r_1,t_2) \psi_{R_2}(c,t_1,t_2,s_2)$$  \hspace{1cm} (3.29)$$

and

$$\phi_{T_2}(c,t_1,r_1) = \max_{t_2 \in T_2} \phi_{R_2}(c,t_1,r_1,t_2)$$  \hspace{1cm} (3.30)$$

for any strategy $s_3$ that follows action $t_2$ with strategy $s_2$.

**Eliminating Variable $R_1$**

The observation variable $R_1$ is then eliminated, creating a strategy variable $S_4$ of 4-stage strategies. The strategy set $\bar{S}_4$ initially has 81 strategies, and after pruning the dominated strategies, we get the minimum set $S_4$ that has 9 undominated strategies, and yielding the formula

$$MEU = \max_{t_1 \in T_1} \max_{s_4 \in S_4} \sum_{c \in C} P(c) \phi_{R_1}(c,t_1) \psi_{R_1}(c,t_1,s_4)$$  \hspace{1cm} (3.31)$$

based on the equivalence:

$$\max_{s_4 \in S_4} \psi_{R_1}(c,t_1,s_4) = \sum_{r_1 \in R_1} \max_{s_3 \in S_3} P(r_1|c,t_1) \phi_{T_2}(c,t_1,r_1) \psi_{T_2}(c,t_1,r_1,s_3).$$  \hspace{1cm} (3.32)$$

and

$$\phi_{R_1}(c,t_1) = \sum_{r_1 \in R_1} P(r_1|c,t_1) \phi_{T_2}(c,t_1,r_1)$$  \hspace{1cm} (3.33)$$
Figure 3.1

Undominated strategies for the used car buyer ID, before selection of the optimal strategy for the initial belief.

Eliminating Variable $T_1$

Then the decision variable $T_1$ is eliminated, creating a strategy variable $S_5$ of 5-stage strategies. The strategy set $S_5$ initially has 36 strategies, and after pruning the dominated strategies, we get the minimum set $S_5$ that has only 4 undominated strategies. Notice that, if dominated strategies are not removed at each step, over 34 million different 5-stage strategies would be generated! We update the MEU as follow:

$$MEU = \max_{s_5 \in S_5} \sum_{c \in C} p(c) \phi_{T_1}(c) \psi_{T_1}(c, s_5),$$

based on

$$\max \psi_{T_1}(c, s_5) = \max_{s_5 \in S_5} \max_{t_1 \in T_1} \phi_{R_1}(c, t_1) \psi_{R_1}(c, t_1, s_4).$$

and

$$\phi_{T_1}(c) = \max_{t_1 \in T_1} \phi_{R_1}(c, t_1).$$
Figure 3.1 shows the set of undominated strategies created by the new algorithm before
the optimal strategy for the initial belief state is selected. To avoid confusion caused by
too many arcs in the graph, for oval nodes corresponding to observations, we show a list
of indices to the successor node, where the position of the index in the list corresponds to
the observation, and the order of observations is shown at the top of the column of nodes.

Eliminating Variable C

Finally, the unobservable variable C is eliminated, creating a strategy variable S_6 of
6-stage strategies. The strategy set \( \bar{S}_6 \) initially has 4 strategies, and after pruning the dom-
inated strategies, we get the minimum set \( S_6 \) that has only 1 undominated strategies. and
yielding the formula:

\[
MEU = \max_{s_6 \in S_6} \psi_C(s_6),
\]

(3.37)

based on

\[
\max_{s_6 \in S_6} \psi_C(s_6) = \max_{s_5 \in S_5} \sum_{c \in C} \phi_T(c) \psi_T(s_5, s_6).
\]

(3.38)

At this point, the problem is solved. We want the strategy that have the maximum value:

\[
\max_{s_6 \in S_6} \psi_C(s_6),
\]

since there is only one strategy in \( S_6 \), that strategy is the solution of this
problem.

Traditional Elimination Ordering

Although our VE algorithm allows a more flexible order of eliminations than the tra-
ditional VE algorithm, it can also eliminate variables in the same order as the traditional
algorithm. The traditional elimination ordering is: \( C \rightarrow B \rightarrow R_2 \rightarrow T_2 \rightarrow R_1 \rightarrow T_1 \),
that is, if it eliminates all hidden variables before eliminating the decision variables and
observation variables. In this case, it performs exactly the same computations as the tra-
ditional algorithm, the only difference is that the new algorithm uses strategy variables in
the computation and explicitly represents the strategy as a graph, and the strategy variable
may still refer to multiple undominated strategies, but there will also never be more than
one undominated strategy for the same instantiation of history. let us verify that statement
by working through the used car buyer example again.

Initialization Step

The initialization step is the same, we substitute \( U(c, t_1, t_2, b) \) with \( \psi(c, t_1, t_2, b, s_0) \),
and get Equation (3.22):

\[
MEU = \max_{t_1 \in T_1} \sum_{r_1 \in R_1} \max_{t_2 \in T_2} \sum_{r_2 \in R_2} \left( \max_{b \in B} \sum_{s_0 \in S_0} \sum_{c \in C} \right) \left( P(c) P(r_1 | c, t_1) P(r_2 | c, t_1, t_2) \psi(c, t_1, t_2, b, s_0) \right).
\] (3.39)

Eliminating Variable \( C \)

First, the unobservable variable \( C \) is eliminated, based on generalized sum-
marginalization,

\[
MEU = \max_{t_1 \in T_1} \sum_{r_1 \in R_1} \max_{t_2 \in T_2} \sum_{r_2 \in R_2} \left( \max_{b \in B} \sum_{s_1 \in S_1} \max_{\phi_C(t_1, r_1, t_2, r_2)} \psi_C(t_1, r_1, t_2, r_2) \psi_C(t_1, r_1, t_2, r_2, b, s_1) \right). \] (3.40)

based on

\[
\phi_C(t_1, r_1, t_2, r_2) = \sum_{c \in C} \tilde{P}(c) P(r_1 | c, t_1) P(r_2 | c, t_1, t_2)
\]

and

\[
\psi_C(t_1, r_1, t_2, r_2, b, s_1) = \sum_{c \in C} P(c) P(r_1 | c, t_1) P(r_2 | c, t_1, t_2) \psi(c, t_1, t_2, b, s_0) \phi_C(t_1, r_1, t_2, r_2).
\]

Notice that, \( \psi_C(t_1, r_1, t_2, r_2, b, s_1) \) represent a scalar value for for each instantiation of its
input variables, just like the traditional VE algorithm, since there is not hidden variable
include in its domain. Moreover, because it is a scalar value, $\delta_1(h)$ can only have one strategy included at most. In other words, $s_1$ become useless and $\psi_C(t_1, r_1, t_2, r_2, b, s_1) = \psi_C(t_1, r_1, t_2, r_2, b)$. Since this situation always happens in the traditional VE, they omit $s_1$ and represent the strategy variable implicitly. But we represent the strategy variable explicitly using $s_1$.

Another thing we need to pay attention is that, even for each history instantiation $h$ there is only one strategy, there are more histories we need to considerate. Consider $\psi_B(s, t_1, t_2, s_1)$ in Equation (3.24), it only has $\{t_1, t_2\}$ as history variables, but $\psi_C(t_1, r_1, t_2, r_2, b, s_1)$ in Equation (3.40), it has $\{t_1, r_1, t_2, r_2, b\}$ as history variables.

**Eliminating Variable $B$**

The decision variable $B$ is eliminated next, leading to creation of a set $S_2$ of 2-stage strategies, and the formula:

$$MEU = \max_{t_1 \in T_1} \max_{r_1 \in R_1} \max_{t_2 \in T_2} \max_{r_2 \in R_2} \max_{s_2 \in S_2} \phi_C(t_1, r_1, t_2, r_2) \psi_B(t_1, r_1, t_2, r_2, s_2).$$

based on:

$$\psi_B(t_1, r_1, t_2, r_2, s_2) = \max_{b \in B} \psi_C(t_1, r_1, t_2, r_2, b, s_1).$$

Again, $\delta_2(h)$ can only have one strategy included at most. Therefore, $s_2$ become useless and $\psi_B(t_1, r_1, t_2, r_2, s_2) = \psi_B(t_1, r_1, t_2, r_2)$.

**Eliminating Variable $R_2$**

The observed chance variable $R_2$ is eliminated next, leading to creation of a set $S_3$ of 3-stage strategies, and the formula:

$$MEU = \max_{t_1 \in T_1} \sum_{r_1 \in R_1} \left( \max_{t_2 \in T_2} \max_{s_2 \in S_2} \phi_{R_2}(t_1, r_1, t_2) \psi_{R_2}(t_1, r_1, t_2, s_2) \right).$$
based on:

\[ \phi_{R_2}(t_1, r_1, t_2) = \sum_{r_2 \in R_2} \phi_C(t_1, r_1, t_2, r_2) \]

and

\[ \psi_{R_2}(t_1, r_1, t_2, s_3) = \sum_{r_2 \in R_2} \left( \frac{\phi_C(t_1, r_1, t_2, r_2) \psi_B(t_1, r_1, t_2, r_2, s_2)}{\phi_{R_2}(t_1, r_1, t_2)} \right) \]

**Eliminating Variable** \( T_2 \)

The decision variable \( T_2 \) is eliminated next, leading to creation of a set \( S_4 \) of 4-stage strategies, and the formula:

\[ MEU = \max_{t_1 \in T_1} \sum_{r_1 \in R_1} \left( \max_{s_4 \in S_4} \phi_{T_2}(t_1, r_1) \psi_{T_2}(t_1, r_1, s_4) \right) \]

based on:

\[ \psi_{T_2}(t_1, r_1, s_4) = \max_{t_2 \in T_2} \phi_{R_2}(t_1, r_1, t_2) \psi_{R_2}(t_1, r_1, t_2, s_3) \]

and

\[ \phi_{T_2}(t_1, r_1) = \max_{t_2 \in T_2} \phi_{R_2}(t_1, r_1, t_2) \]

**Eliminating Variable** \( R_1 \)

The observed chance variable \( R_1 \) is eliminated next, leading to creation of a set \( S_5 \) of 5-stage strategies, and the formula:

\[ MEU = \max_{t_1 \in T_1} \max_{s_5 \in S_5} \phi_{R_1}(t_1) \psi_{R_1}(t_1, s_5) \]

based on:

\[ \phi_{R_1}(t_1) = \sum_{r_1 \in R_1} \phi_{T_2}(t_1, r_1) \]

and

\[ \psi_{R_1}(t_1, s_5) = \sum_{r_1 \in R_1} \left( \frac{\phi_{T_2}(t_1, r_1) \psi_{T_2}(t_1, r_1, s_4)}{\phi_{R_1}(t_1)} \right) \]
Eliminating Variable $T_1$

The decision variable $T_1$ is eliminated next, leading to creation of a set $S_6$ of 6-stage strategies, and the formula:

$$MEU = \max_{s_6 \in S_6} \psi_{T_1}(s_6)$$

based on:

$$\psi_{T_1}(s_6) = \max_{t_1 \in T_1} \phi_{R_1}(t_1) \psi_{R_1}(t_1, s_5)$$

Finally, we pick the strategy $s_6 \in S_6$ that will have the maximum expected utility.

3.3 Elimination Ordering Heuristics

A key difference between our algorithms and the traditional algorithms for evaluating IDs is that our algorithms allow removing decision and observed variables before removing unobservable chance variables since our algorithms have the ability that can reasoning on the hidden states. This difference relaxes the elimination ordering constraints when evaluating IDs, and allow a more flexible elimination ordering. The elimination order has a huge effect on the time complexity of both our generalized algorithms and traditional ID algorithms. However, finding the optimal elimination ordering is an NP-hard problem [29]. There are several existing heuristics that have been proposed for traditional variable elimination algorithm, such as Minimum size [49], Minimum weight [29] and Cano and Moral [9]. We proposed a heuristic that is similar to the Minimum weight heuristic but also can take the advantage the more flexible elimination ordering to improve the GVE’s efficiency.
For problems that the number of histories is extremely large and the size of optimal strategy is relatively small, reasoning on the belief state’s approach is effective, and we would like to have an elimination order that uses this approach. In fact, the maze navigation problem provides both a best-case of the belief state’s approach and a worst-case of the history-based approach.

On the other hand, the mildew treatment problem provides a case that the history-based approach is better than the belief state approach since the size of the undominated strategies for any belief states is huge but the number of the total histories is small. For problems like this, it is better to apply an elimination order that uses the history-based approach instead of the belief state approach. Fortunately, our generalized algorithm has the ability to execute both approaches and which approaches the algorithm will execute is only depending on the elimination order.

Before we introduce the elimination ordering heuristic for solving ID, we need to understand that when using the belief state approach, the growth in the number of strategies from one stage to the next is problem-dependent and much less predictable. The dynamic programming approach that is used in belief state approach is an output-sensitive algorithm. In other words, we don’t know the size of the optimal strategies without solving the problem. As a result, the heuristic we proposed dynamically sets the eliminate order as the algorithm runs, in contrast to the current practice, where it is typical to use a static elimination ordering that is determined before the start of the algorithm.

Intuitively, we want our elimination ordering heuristic to be able to do the following two things: First, when the size of the current strategies is small, typically happens in the
last of couple stages, the heuristic would like to choose a variable to eliminate, when this variable is eliminated, the size of the strategies will increase, but the number of the histories won’t. Second, when the size of the current strategies is big but the number of histories is small, typically happens in the first of couple stages, the heuristic would like to choose a variable to eliminate (for example, hidden variables), when this variable is eliminated, the number of the histories will increase, but the size of the strategies will decrease.

The time complexity of the traditional variable elimination algorithm is polynomial time to the size of the potential during the evaluation. However, the size of the potential grows exponentially to the relevant variables in the domain of the potential. Therefore, the size of the potential can be used as a factor to determine our elimination ordering heuristic. The size of the potential can be considered as the weight of the clique candidate that is used in the Minimum weight heuristic [29]. We use $N_i$ to represent the size of utility potential that generated by eliminate variable $i$ (stage $i$) and we have:

$$N_i = C(X_i) \cdot C(H_i)$$

where $C(X_i)$ is the Cartesian product of all hidden variables $X_i$ that in $\psi_i$’s domain and $C(H_i)$ is the Cartesian product of all history variables $H_i$ that in $\psi_i$’s domain. We use $O_t$ to denote the size of the biggest utility potential that occurs when evaluating the IDs using traditional variable elimination algorithm and some elimination ordering, and we have:

$$O_t = \max_i N_i$$

76
The time complexity of the generalized variable elimination is also polynomial time to the size of the potential during the evaluation. We use $M_i$ to represent the size of utility potential that generated by eliminate variable $i$ (stage $i$) and we have:

$$M_i = C(X_i) \cdot C(H_i) \cdot |S_i|$$

where $C(X_i)$ is the Cartesian product of all hidden variables $X_i$ that in $\psi_i$’s domain, $C(H_i)$ is the Cartesian product of all history variables $H_i$ that in $\psi_i$’s domain and $|S_i|$ is the number of states of strategy variable $S_i$ (the number of strategies). We use $O_g$ to denote the size of the biggest utility potential that occurs when evaluating the IDs using generalized variable elimination algorithm with and some elimination ordering, and we have:

$$O_g = \max_i M_i$$

The goal of the heuristic is by choosing some elimination order order 1, that only allows in the generalized VE algorithm, in order to get $O_g < O_t$. And the elimination order heuristic have the following rules:

1. The elimination order should always satisfy the elimination ordering constrains for GVE.

2. If remove a hidden variable $X_i$ in stage $i$ will not increase either $C(X_{i+1})$ or $C(H_{i+1})$ for stage $i + 1$ (last stage’s utility potential size), then we should always remove that hidden variable. (remove hidden variable won’t increase $|S_i|$ over $|S_{i+1}|$.)

3. If remove a hidden variable $X_i$ in stage $i$ will increase $C(X_{i+1})$ or $C(H_{i+1})$ for stage $i + 1$, then we are facing a choose that is are we still want to remove this hidden variable $X_i$, or we want to remove another decision or observation variable $Y_i$ that will decrease
Although removing a decision or an observation variable $Y_{i+1}$ will decrease $C(H_{i+1})$, it will also potentially increase $|S_{i+1}|$ or $C(X_{i+1})$. Therefore, we follow a greedy approach to determine either to remove $X_i$ or $Y_i$ as follow:

$$\min \left( M_i^{X_i}, M_i^{Y_i} \right)$$

(3.41)

where $M_i^{X_i}$ is the size of the resulting potential if remove $X_i$ and $M_i^{Y_i}$ is the size of the resulting potential if remove $Y_i$. We are going to remove the variable that will give us a smaller size.

For a given ID problem, $C(X_i)$ and $C(H_i)$ for each stage $i$ can only be changed when applying different elimination ordering. In other words, once the elimination ordering is fixed, we can calculate $C(X_i)$ and $C(H_i)$ for each stage a head of time without solving the problem, and this give us a baseline $M_i$ for comparison. However, $|S_i|$ is depend on how complex the real strategy is of the problem, and that is why the algorithm needs to determine which variable is going to remove dynamically during the evaluation process.

Moreover, we can also control $|S_i|$ by introduce an error threshold $\epsilon$. The approximation method that we are going to discuss is a bounded-error approximation method, and we can get upper bound from this approach. (The algorithm is smartly choosing the more important strategies to keep, and forget less important strategies based on the expected utility)

### 3.4 Bounded-Error Approximation

Although exact algorithms give the optimal solution to the problems, they may not be able to solve very large problems. We will introduce an approximately method that will
improve the scalability of our new approach for solving IDs. As we discussed before, the size of the hidden states, the size of the histories and the number of the undominated strategies are the three source of complexity in our new approach since it integrated IDs and POMDPs solving techniques. We will focus on using approximate methods to reduce the number of the undominated strategies and then lead to improving the efficiency of the algorithm. More specifically, we want to modify an existing approximate method for factored POMDPs that first proposed by Feng and Hansen [18, 21] to fit in our new approach for solving IDs.

As we introduced before, both incremental pruning algorithm for POMDPs and generalized variable elimination algorithm for IDs execute Lark’s algorithm as a subroutine to eliminate dominated vectors in the vector set $\mathcal{V}$. The difference between incremental pruning algorithm and GVE algorithm is how to set up Lark’s algorithm (determine the input parameter). Once the input parameter, a set of $|X|$-dimensional vectors $\mathcal{V}$, is determined, the execution of Lark’s algorithm is exactly the same. Therefore, the bounded-error approximate technique we proposed only need to modify Lark’s algorithm.

The basic idea of the approximation technique is to eliminate $\epsilon$-dominated strategies, which are strategies with a value that is not better that the value of the best alternative strategy by more than a threshold $\epsilon$. Algorithm 9 shows the modified Lark’s algorithm. Beside the set of vectors $\mathcal{V}$, we also have a error threshold $\epsilon$ as an input parameter. In line 6 of Algorithm 9, we see, a vector is considered non dominated when $d - \epsilon > 0$ instead of $d > 0$. Therefore, with the larger $\epsilon$, more strategies will be eliminated, the resulting strategy set will be smaller, and the algorithm will be faster. However, the solution quality will be
Algorithm 9: Approximate Lark’s algorithm

Input: A non-minimal set of vectors \( \mathcal{V} \) representing a value function, a error threshold \( \epsilon \)

Output: A randomized minimal set of vectors \( \mathcal{V} \) representing the value function with maximum \( \epsilon \) decrease

1. \( \mathcal{V} \leftarrow \emptyset \)
2. Prune all pointwise dominated vectors from \( \mathcal{V} \)
3. while \( \mathcal{V} \neq \emptyset \) do
   4. Pick one \( v \in \mathcal{V} \)
   5. \( \{d, b(s)\} \leftarrow \text{LP}(v, \mathcal{V}) \)
   6. if \( d - \epsilon > 0 \) then
      7. \( v^* \leftarrow \max_{v \in \mathcal{V}} (v \cdot b(s)) \)
      8. remove \( v^* \) from \( \mathcal{V} \)
      9. add \( v^* \) into \( \mathcal{V} \)
   10. else
      11. remove \( v \) from \( \mathcal{V} \)
   12. end
4. return \( \{\text{A minimal set of vectors } \mathcal{V}\} \)

lower. Notice that, once we introduce approximation in Lark’s algorithm, the minimal set \( \mathcal{V} \) is not unique anymore, since the minimal set \( \mathcal{V} \) can represent the value between the original value \( |\mathcal{V}| \) and \((|\mathcal{V}| - \epsilon)\), and also creates an error that is bounded by \( \epsilon \).

3.5 Experimental Results

We use three test problems to illustrate the improved scalability and the resulting compacted strategy graph of our new approaches.

Maze navigation Problem

Figure 3.2 shows four partially observable maze navigation problems introduced in previous work on limited-memory IDs [42]. The shaded tiles represent walls, the white tiles represent movable space, and the white tile with a star is the goal state. A robot in the maze has four available actions; it can move a single step in any of the four compass
Solving Influence Diagrams Using Heuristic Search

Changhe Yuan & Xiaojian Wu
Department of Computer Science and Engineering
Mississippi State University
Mississippi State, MS 39762
chuan@cse.msstate.edu & xw83@msstate.edu

Abstract
Existing methods for solving influence diagrams are mostly based on the bottom-up dynamic programming technique. These methods may waste computation in solving decision scenarios that have zero probabilities or are unreachable from any initial state by following an optimal decision policy. Heuristic search was applied in (Qi & Poole 1995) to address these limitations. However, their algorithm uses a trivial infinity upper bound and fails to fully utilize the potential of heuristic search. This paper develops an improved heuristic search algorithm for solving influence diagrams based on a more informative upper bound computed by relaxing the models. The algorithm is shown to be able to significantly improve the efficiency and scalability of existing methods for solving influence diagrams.

1 Introduction
An influence diagram (Howard & Matheson 1981) provides a compact and intuitive representation of the relationships between random variables, decisions, and preferences in a domain and offers a popular framework for decision making under uncertainty. Numerous algorithms have been developed to solve influence diagrams (Bhattacharya & Shachter 2007; Jensen, Jensen, & Dittmer 1994; Shenoy 1992; Zhang 1998; Qi & Poole 1995; Cooper 1988; Shachter 1986; Olmsted 1983). Most of these algorithms, whether they build a secondary structure or not, are based on the bottom-up dynamic programming approach. They start by solving small low-level decision problems and gradually build on the results to solve larger problems until the solution to the global-level decision problem is found. The drawback of these methods is that they may waste computation in solving decision scenarios that have zero probabilities or are unreachable from any initial state by following an optimal decision policy. Qi and Poole propose to address this problem using heuristic search (Qi & Poole 1995), but their method uses the trivial infinity upper bound to guide the search. A more informative upper bound needs to be developed to fully utilize the potential of heuristic search.

This paper develops an improved heuristic search algorithm for solving influence diagrams. We first note that, for a partially observable domain, we can build an upper-bound influence diagram that not only is simpler but also provides an upper bound for the maximum expected utility of the original influence diagram (Nilsson & Hohle 2001). We then use a strong jointree of the upper-bound influence diagram to compute the probabilities and upper-bound utilities needed in an AND/OR graph search algorithm to find an optimal policy graph for the original influence diagram. Our empirical results show that the new algorithm can significantly improve the efficiency of the state-of-the-art solution methods for influence diagrams on large decision problems.

Maze

Figure 1: Four maze domains.

Figure 3.2

Four Different Mazes

Maze

Figure 3.3

Influence diagram of the maze navigation problem (graphical part).

Section, which can sense whether there is wall in the neighboring tile of that direction. However, the sensors are noisy. It can detect wall correctly with probability 0.9, and it may give false alarm with probability 0.05. Based on the results of the sensors, the agent decides whether to move to a neighboring tile. The action is also noisy. The agent may fail to
directions. The robot successfully moves in the intended direction with probability 0.89. It fails to move with probability 0.089, it moves sideways with probability 0.02 (0.01 for each side), and it moves backward with probability 0.001. If movement in some direction would take it into a wall, that movement has probability zero, and the probability of not moving is increased. The robot has four sensors, one for each direction, which accurately sense whether the neighboring tile in that direction is a wall. Because different states can result in the same observation, the problem is partially observable. Figure 3.3 shows the graphical part of the influence diagram of this problem, and for the space reason, it only shows parts of the entire diagram. Notice that, the non-forgetting arcs are also omitted to make the diagram clear but we do consider them when evaluating the problem.

A robot is randomly placed in a non-goal state (i.e., the prior probability distribution is uniform), and performs an action at each of a sequence of ten stages. If it reaches the goal state by the final stage, it receives a reward of 1; otherwise, it receives a reward of 0. Thus the objective is to maximize the probability of reaching the goal state within ten stages.

Although [42] model the problem as an ID, it is a very challenging problem for traditional ID algorithms to solve. At each stage, the robot receives one of 16 observations and performs one of 4 actions, which means that $64^{10}$ different histories are possible over ten stages. Given so many histories, traditional algorithms for ID evaluation cannot solve this problem. Even representation of a strategy as a simple mapping from histories to actions is infeasible.

Surprisingly, Figure 3.2 (a) can be solved in less than a minute by our generalized variable elimination algorithm, if it adopts an variable elimination ordering that reflects the
POMDP approach and allows hidden state at each stage. We use the following variable elimination ordering. We can consider this problem have 10 parts, and let \( i \) starts with 1. After finish each part, \( i \) increase by 1 until 10. For each part, we remove all the hidden variables in \( U_i \) first, all the observation variables in \( I_i \) second, and then remove the decision variable \( d_i \). With this variable elimination ordering, the algorithm performs the same sequence of computations as the incremental pruning algorithm. The maze (a) problem is easily solved by incremental pruning because an optimal strategy for this problem can be represented compactly by a strategy graph (we will discuss the details of how to construct the strategy graph in chapter 5) that shows in Figure 3.4 with only 44 action (OR) nodes.

![Maze Strategy Graph](image)

**Figure 3.4**

Maze Strategy Graph

The other mazes turn out to have different degrees of difficulty. Maze (b) is solved by our algorithm in 9 minutes. It has 1366 undominated 10-stage strategies before considering the initial belief state. Maze (c) cannot be solved by our algorithm in less than half an hour of CPU time; the last 8 stages of the problem are solved in seven and a half minutes, and
there are 2999 undominated 8-stage strategies. Maze (d) is by far the easiest to solve. It takes less than 30 CPU seconds to solve and there are only 4 undominated 10-stage strategies.

**Mildew treatment**

![Diagram of Mildew treatment](image)

**Figure 3.5**

Mildew problem influence diagram (a) and strategy graph (b)

We consider a simple ID that models the problem of a farmer deciding how to eliminate mildew in a wheat field by selecting among fungicide treatments. The example is originally described by [26, pp. 282-283], with more details given at the HUGIN website.¹

As shown in Figure 3.5 (a), the ID has one decision node (A) with four options for fungicide treatment (none, light, moderate, and heavy). It has two observation nodes: observation of the crop state (OQ), with 4 possible values (fair, average, good, and very good), and observation of the mildew situation (OM), with 4 possible values (none, little, moderate, and severe). It has four unobservable chance nodes. The initial crop state (Q)

---

has 4 values. The crop state at harvest (H) has 7 values. Both the mildew situation before
treatment (M) and the mildew situation after treatment (M*) have 4 values.

Two of the hidden variables must be removed before the decision variable is removed. The
other hidden variables can be removed either before the observation variables are
removed or after the observation variables are removed. If they are removed before the
observation variables are removed, the problem is solved in a fraction of a second by
our algorithm. (Note that this variable elimination ordering is the same order in which
the nodes would be removed by the traditional variable elimination algorithm.) If the
hidden variables are not removed until all of the other variables are removed, including the
observation variables, the problem cannot be solved within 30 minutes by our algorithm
because of the large number of undominated strategies.

For this problem, it is more effective to remove the hidden variables before removing
the two observation variables because only 16 histories need to be considered to solve
this problem, whereas the number of possible strategies is over 4 billion, and many of
them are undominated. Although our algorithm does not solve this problem any faster
than the traditional variable elimination algorithm, it represents the optimal strategy more
compactly, as shown in Figure 3.5 (b).

**The ALARM Monitoring System problem**

We next consider an ID that is based on a Bayesian network called ”A Logical Alarm
Reduction Mechanism” (ALARM) [2] that is a diagnostic application used to explore prob-
abilistic reasoning techniques in belief networks. The goal of the ALARM monitoring
system is to provide specific text messages advising the user of possible problems.
Figure 3.6
The Influence Diagram of the ALARM Monitoring System Problem
We transform this Bayesian network into an ID by adding a decision variable that represents a choice of two possible treatments, as well as a utility node that is a function of both the underlying diseases and the treatments. The ID is shown in Figure 3.6. The decision and utility nodes that we added are not intended to be medically realistic; they were added simply to create a useful test problem.

The ALARM ID problem has 16 observed chance variables, 19 hidden chance variables, and one decision variable that have two actions. After removing all the hidden chance variable, all the 16 observed chance variables and the decision variable are relevant to the utility. Therefore, the total number of relevant histories is 45,349,632, which is too big to solve by traditional influence diagram algorithm. Therefore, \( O_t = C(X_i) \cdot 45,349,632. \)

If using our GVE with an error \( \epsilon = 1 \) and apply the elimination ordering order1, the \( O_g = \max_i C(X_i) \cdot C(H_i) \cdot |S_i|, \) where \( C(H_i) = 72 \) and \( |S_i| = 511. \) Therefore, we have \( O_g = C(X_i) \cdot 36,792. \) For this problem, we can make sure \( C(X_i) \) is same in both traditional approach and new approach. Notice that, \( O_g \) is 1,232 times smaller than \( O_t, \) and this is also the reason of why the GVE can solve this problem.
CHAPTER 4
THE GENERALIZED ARC REVERSAL ALGORITHM

In this chapter, we are going to introduce how to integrate POMDPs techniques with traditional Arc-Reversal algorithm when evaluating influence diagrams. We name our new approach Generalized Arc Reversal algorithm [22]. Let us introduce the background of traditional Arc-Reversal algorithm first.

4.1 Arc-Reversal Algorithm

Arc-Reversal [51] is the first algorithm that evaluates ID directly on the ID itself. The algorithm removes a single node each iteration using one of the five value-preserving reductions and transforms the ID into a smaller ID until the only node left is the value node. A reduction is called value-preserving if and only if such a reduction/transformation do not modify the optimal policy or maximal expected value. First, we will introduce the five value-preserving reductions one by one, then discuss how the algorithm performs. Let X denotes chance nodes, D denotes decision nodes and V denotes value nodes.

Barren node removal

A decision or chance node is said to be barren if it has no children, in which case it can be removed without having any effect on the value of an ID [51]. Figure 4.1 shows an example that removes barren nodes from an influence diagram. Once a barren node has
been removed, other nodes may become a barren node. For example node A becomes a barren node after node B has been removed. Therefore, we need to remove node A as well.

**Value node removal**

Although the original node-removal/arc-reversal algorithm assumed a single value node, it was later extended to allow multiple value nodes. When there are multiple value nodes, any two value nodes can be replaced by a single value node that is equal to their sum \[59\]. Let \(U_{old}(pa(U))\) denote the value function associated with the value node \(U\), and let \(V_{old}(pa(V))\) denote the value function associated with the value node \(V\) before the two
value nodes are merged. The value function associated with the new value node $V_{\text{new}}$ that replaces the two merged value nodes is

$$V_{\text{new}}(c) = U_{\text{old}}(a) + V_{\text{old}}(b),$$

where $A = pa_{\text{old}}(U)$ is the original set of parents of $U$, $B = pa_{\text{old}}(V)$ is the original set of parents of $V$, and $C = pa_{\text{new}}(V) = A \cup B$ is the new set of parents of $V$. Figure 4.2 shows the changes of an ID before and after removing the value node $U$.

To gain computational leverage from decomposition of the value function, it is well-

A chance node $X$ can be removed if it has a single child value node [51, pp. 876-7]. Let $P_{\text{old}}(X|pa(X))$ denote the conditional probability table associated with a chance node $X$, and let $V_{\text{old}}(pa(V))$ denote the value function associated with its child value node $V$. 

![Figure 4.3](image.png)

**Figure 4.3**

Chance Node Removal
before removing $X$. After removing $X$, the parents of $X$ are inherited by the value node $V$, and the new value function associated with $V$ is

$$V_{\text{new}}(x) = \sum_{x \in X} P_{\text{old}}(x|a) \cdot V_{\text{old}}(b),$$  \hspace{1cm} (4.2)$$

where $A = \text{pa}_{\text{old}}(X)$ is the original set of parents of $X$, $B = \text{pa}_{\text{old}}(V)$ is the original set of parents of $V$, and $C = \text{pa}_{\text{new}}(V) = A \cup (B \setminus \{X\})$ is the new set of parents of $V$.

![Diagram showing decision node removal](image)

**Figure 4.4**

Decision Node Removal

When a decision node $D$ has only one child, which is a value node $V$, and all other parents (conditional predecessors) of the value node are also parents (informational predecessors) of the decision node, the decision node may be removed, and the new value function associated with the value node is

$$V_{\text{new}}(b) = \max_{d \in D} V_{\text{old}}(a) = \max_{d \in D} V_{\text{old}}(d, b),$$  \hspace{1cm} (4.3)$$
where \( A = pa_{old}(V) \), and \( B = pa_{new}(V) = pa_{old}(V) \setminus \{ D \} \) is the new set of parents of \( V \). The optimal decision \( d \) for each instantiation \( b \) is stored as the optimal policy for \( D \).

Figure 4.4 shows the changes of an ID before and after remove the decision node \( D \).

Note that when a decision node \( D \) that precedes a value node \( V \) is removed, any parents of \( D \) that are not also parents of \( V \) are ignored in choosing the optimal policy and might become barren. The parents of \( D \) that are also parents of \( V \) are the requisite observations.

![Figure 4.5](image.png)

**Figure 4.5**

Reverse the arc between two chance nodes

Arc reversal uses Bayes’ rule to reverse the direction of an arc between two chance nodes, while ensuring that the original distribution is still correctly represented [51, pp. 878]. An arc from a chance node \( X \) to a chance node \( Y \) with parent sets \( A \) (parents of \( X \) only), \( B \) (parents of both \( X \) and \( Y \)), and \( C \) (parents of \( Y \) only) can be reversed unless there is another directed path from \( X \) to \( Y \); in that case, reversing the arc would create a directed cycle.
When an arc \((X, Y)\) can be reversed, the new distribution for \(Y\) is obtained by marginalizing \(X\) from the joint, as follows:

\[
P_{\text{new}}(y|a, b, c) = \sum_{x \in X} P_{\text{old}}(y|x, b, c) P_{\text{old}}(x|a, b).
\]

(4.4)

Since the chance node \(Y\) inherits the parents of \(X\), its new parents are \(\text{pa}_{\text{new}}(Y) = \text{pa}_{\text{old}}(Y) \cup \text{pa}_{\text{old}}(X) = A \cup B \cup C\).

The new distribution for \(Y\) is used to obtain the new distribution for \(X\), as follows:

\[
P_{\text{new}}(x|y, a, b, c) = \frac{P_{\text{old}}(y|x, b, c) P_{\text{old}}(x|a, b)}{P_{\text{new}}(y|a, b, c)}.
\]

(4.5)

Since the chance node \(X\) is now a child of \(Y\), and also inherits the parents of \(Y\), its new parents are \(\text{pa}_{\text{new}}(X) = \{Y\} \cup \text{pa}_{\text{old}}(X) \cup \text{pa}_{\text{old}}(Y) = \{Y\} \cup A \cup B \cup C\). Figure 4.5 shows the changes of an ID before and after reverse chance nodes \(X\) and \(Y\).

**Arc Reversal algorithm pseudocode**

Algorithm 10 shows the pseudocode of the Arc-Reversal algorithm. It takes an influence diagram as input, and then remove a single node in the diagram that satisfied the removable criterion until the only node left is the terminate value node \(V_t\). The value of the \(V_t\) is the maximum expected utility of the ID. There are two removable criterions, one for determining if a chance node can be removed or not, and another one for determining if a decision node can be removed or not, and they are:

**Definition 4 (Removable Chance Node)**

*If \(X\) is a chance node in an influence diagram with no decision successors. Then \(X\) can be removed from the diagram.*
Algorithm 10: Arc Reversal Algorithm

**Input:** An Influence Diagram
**Output:** Maximum Expected Utility for the Influence Diagram

1. Add “no-forgetting” arcs
2. Eliminate any barren nodes
3. Initialize super value node $V_t$
4. while $pa(V_t) \neq \emptyset$ do
   5. if there is a removable decision node $D_i$ then
      6. Apply value node removal on all successor value nodes that are not $V_t$
      7. Apply decision node removal on $D_i$
      8. Eliminate any resulting barren nodes
   else
      /* there must be a removable chance node $X_i$*/
      10. if there is a chance node $X_i$ that all of its successors are value nodes then
          11. Apply value node removal on all successor value nodes that are not $V_t$
          12. Apply chance node removal on $X_i$
      else
          15. while the chance node $X_i$ has another chance node $X_j$ do
             16. Apply Arc reversal between $X_i$ and $X_j$
          end
          18. Apply value node removal on all successor value nodes that are not $V_t$
          19. Apply chance node removal on $X_i$
      end
5. end
6. return $\{V_t\}$

Definition 5 (Removable Decision Node)

If $D$ is a decision node in an influence diagram with terminal value node $V_t$ such that:

a) There is no chance node is a successor of $D$.

b) All the conditional predecessor of $V_t$, beside $D$, are informational predecessor of $D$.

4.2 Generalized Arc Reversal Algorithm

In this section, we are going to introduce a node-removal/arc-reversal [51, 59] algorithm for influence diagram evaluation that includes reductions that allow an influence diagram to be solved by a generalization of the dynamic programming approach to solv-
ing partially observable Markov decision processes (POMDPs), called *Generalized Arc Reversal*.

The algorithm we describe next can be viewed as a synthesis of the incremental pruning algorithm and the classic node-removal/arc-reversal algorithm for ID evaluation. Briefly, the algorithm is created by generalizing the cross sum and maximization steps of incremental pruning to create new reductions that apply to the observed chance nodes and the decision nodes of an ID, respectively, and replacing the backprojection step of the incremental pruning algorithm with reductions used by the node-removal/arc-reversal algorithm.

### 4.2.1 Strategy Node and Strategy Value Node

Our new algorithm first solves an ID for the last stage of the problem, then for the last two stages, then for the last three stages, and so on, in a way that generalizes the incremental pruning algorithm. It uses two nodes of the ID to hold the solution as it is gradually constructed. The first node, which we call the *strategy node*, is the last decision node in the ID. It is used to hold a set $S_i$ of strategies, as defined by Equations (3.13) and (3.14). The second node, which we call the *strategy value node*, is a child value node of the strategy node. It serves as a piecewise-linear and concave value function for the set of strategies, as we show below.

The two nodes are originally established in a simple initialization step. First, the last decision node in the ID is identified. Then the standard reductions described above are used to eliminate all of its descendants except for one value node (which must exist if the decision node cannot be removed by a series of barren node removals). Because the
strategy node is just a decision node used in a special way, and the strategy value node is just its child value node, an ID with these two nodes is well-defined.

Let $D$ denote the strategy node, let $V$ denote the strategy value node, let $X$ denote the set of parents of the strategy value node that are unobserved chance nodes, and let $Y$ denote the set of parents of the strategy value node that are observation nodes or decision nodes, excluding the strategy node. For each action $d \in \text{Act}(D)$, and for any instantiation $x \in X$ of the hidden state variables, and for any instantiation $y \in Y$ of the related decision and observation variables, the value function $V(d, x, y)$ associated with the strategy value node gives the value of taking the action $d$. Moreover, for any belief state $b$ over instantiations of the hidden state variables $X$ (that is, for any probability distribution over instantiations $x \in X$), and for any instantiation $y \in Y$ of the related decision and observation variables, the expected value of decision $d$ is given by: $\sum_x b(x)V(d, x, y)$. Finally, for any belief state $b$ over instantiations $x \in X$ of the hidden state variables, and for any instantiation $y \in Y$ of the decision and observation variables, the optimal value function is

$$V(b, y) = \max_{d \in \text{Act}(D)} \sum_{x \in X} b(x)V(d, x, y).$$

(4.6)

As this analysis shows, the value function associated with the strategy value node is a piecewise linear and concave value function, similar to the value function for a POMDP, and we leverage this observation in the algorithm we describe next.

Both the strategy node and the strategy value node play an integral role in the following two new reductions.
Compare to classic Arc-reversal algorithm the generalized Arc Reversal algorithm has two new reductions. The first is new decision node reduction and the second is observation node reduction. When the utility has a parent that is a hidden chance node, the algorithm should always use those two new reductions. Since the other reductions are similar to the classic algorithm with minor changes, we only introduce the two new reductions.

![Diagram of decision node reduction](image)

**Figure 4.6**

Example of ID before (left) and after (right) a decision node reduction. Tables show value function for value node. Shaded strategies in the revised table are dominated.

### 4.2.2 Decision Node Reduction

We first describe a reduction for decision nodes. It can be applied to a decision node if and only if its only children are the strategy node and (possibly) the strategy value node. (It is a parent of both, usually, because if it is not a parent of the strategy value node, it has no effect on value.) Importantly, and by contrast to the decision node reduction for the classic node-removal/arc-reversal algorithm, it allows the child value node of the decision node being reduced to have unobserved chance nodes as parents.
The reduction removes the decision node $X_t$, it modifies the action set associated with the strategy node, and it modifies the value function associated with the strategy value node. Figure 4.6 shows a simple example of the decision node reduction. To indicate the special role played by the strategy node and its value node, the strategy node is shown as a double square labeled $S$, and the strategy value node is shown as a double diamond labeled $V$.

Let $S_{t-1}$ denote the set of $(t-1)$-stage strategies originally associated with the strategy node (in its action set), and let $V_{old}$ denote the value function originally associated with the strategy value node. The reduction replaces the set $S_{t-1}$ with a set $S_t$ of $t$-stage strategies based on Equation (3.13). That is, for each strategy $s_{t-1} \in S_{t-1}$ and for each action $x_t \in \text{Act}(X_t)$ associated with the removed decision node, the new set $S_t$ includes a strategy $s_t = \langle x_t, \text{succ}_t \rangle$.

Let $X$ denote the set of parents of the strategy value node that are unobserved chance nodes, and let $Y$ denote the set of parents of the strategy value node that are either observation nodes or decision nodes, excluding the strategy node and (removed) decision node. When the decision node is removed, the new value function for the strategy value node is:

$$V_{new}(x, y, s_t) = V_{old}(x, y, d_t, s_{t-1}).$$ \hspace{1cm} (4.7)

Note that the sets $X$ and $Y$ are not changed by removing the decision node. Adopting a notation similar to the one we used in the POMDP case, we also let $v^i_t(x, y)$ denote the value vector associated with a strategy $s^i_t \in S_t$. 

98
After the decision node is removed, the last step of the reduction is to eliminate dominated strategies in the new set $S_t$ corresponding to the action set of the revised strategy node. We consider two cases. In the first, $X$ is empty, that is, the strategy value node does not have any parents that are unobserved chance nodes. This case is relatively easy: a strategy $s^i_t \in S_t$ with corresponding value vector $v^i_t$ is dominated if

$$v^i_t(y) \leq \max_{v^k_t \in V_t \setminus \{v^i_t\}} v^k_t(y), \forall y \in Y,$$

which is easily tested by enumerating all instantiations of $Y$.

In the second case, $X$ is non-empty, that is, the strategy value node has parents that are unobserved chance nodes. This case requires reasoning about hidden state in a way that generalizes the pruning step of the incremental pruning algorithm for POMDPs. Given a set $S_t$ of $t$-stage strategies with corresponding value vectors $V_t$, a strategy $s^i_t \in S_t$ is dominated by the other strategies $S_t \setminus \{s^i_t\}$ if for all belief states $b$:

$$\sum_{x \in X} b(x)v^i_t(x, y) \leq \max_{v^k_t \in V_t \setminus \{v^i_t\}} \sum_{x \in X} b(x)v^k_t(x, y), \forall y \in Y.$$

Because this test considers all possible belief states, that is, all probability distributions over the hidden states, it requires use of linear programming. In the worst case, it solves a linear program for each strategy $s^i_t \in S_t$ and instantiation $y \in Y$.

### 4.2.3 Observation Node Reduction

The observation node reduction we describe next can be applied to an observation node if and only if its only children are the strategy node and (possibly) the strategy value node.

The reduction removes the observation node $X_t$, it modifies the action set associated with the strategy node, and it modifies the value function associated with the strategy value.
Examples of observation node reduction. Dominated strategies are shaded in the table for the value function.

node. Again, the algorithm replaces the set $S_{t-1}$ of $(t-1)$-stage strategies originally associated with the strategy node with a new set $S_t$ of $t$-stage strategies based on Equation (3.14).

But for each new strategy $s_t \in S_t$, where $s_t = \langle \text{null}, \text{succ}_t \rangle$.

We must consider how the parent set of the strategy value node is changed by this reduction, and how its new value function is defined. Let $A$ denote the parents of the strategy value node before the reduction that are unobserved chance nodes, and let $B$ denote the parents of the strategy value node before the reduction that are observation or decision nodes, excluding the strategy node and (removed) observation node. Let $C$ denote the parents of the removed observation node that are unobservable chance nodes, and let $D$ denote the parents of the removed observation node that are observation or decision nodes. The new parent set for the strategy value node consists of the unobservable chance nodes.

Figure 4.7
\( X = A \cup C \) and the observation and decision nodes \( Y = B \cup D \). The new value function for the strategy value node is,

\[
V_{\text{new}}(x, y, s_t) = \sum_{z \in \text{dom}(N_t)} P(z|c, d) \cdot V_{\text{old}}(a, b, z, \beta_t(z)), \tag{4.10}
\]

where \( z \in \text{dom}(N_t) \) is an observation, and \( \beta_t(z) \) maps the observation \( z \) to a strategy in \( S^{t-1} \).

Figure 4.7 show some simple examples of this reduction. There is a subtle complication in determining the parent set of the strategy value node after the reduction that we mention to qualify the discussion in the previous paragraph. If a parent node of the observation node is not connected to the strategy value node by a path that does not go through the strategy node, then it is irrelevant in a sense defined by Nielsen [40] and Jensen and Nielsen [26, pp. 413-420], which means it has no effect on expected value. Thus it can be excluded from the parent set of the strategy value node after the reduction. In the example shown in Figure 4.7 (c), for example, the node \( X \) is irrelevant and becomes barren after the reduction.

After the observation node has been removed, the last step of the reduction is to eliminate dominated strategies in the new set \( S_t \) corresponding to the action set of the revised strategy node. Adopting a notation similar to the one used in the POMDP case, we let \( v^i_t(x, y) \) denote the value vector associated with a strategy \( s^i_t \in S_t \). As already explained in the discussion of eliminating dominated strategies after the reduction of a decision node, there are two cases to consider. In the first case, \( X \) is empty, that is, the strategy value node does not have any parents that are unobserved chance nodes. In this case, a strategy \( s^i_t \in S_t \) is dominated if it satisfies the condition of Equation (4.8), which can be tested without lin-
ear programming. In the second case, \( X \) is non-empty, that is, the strategy value node has parents that are unobserved chance nodes. In this case, a strategy \( s^i_t \in S_t \) is dominated by the other strategies \( S_t \setminus \{s^i_t\} \) if it satisfies the test of Equation (4.9) for all belief states \( b \) over the hidden state variables \( X \).

Although there are similarities in how dominated strategies are eliminated for decision nodes and observation nodes, there is also an important difference. But efficient implementation of this pruning step is more challenging for an observation node reduction because the cardinality of the set of strategies grows much more explosively when an observation node is removed than when a decision node is removed. In the incremental pruning algorithm, this step is referred to as the cross sum step, and it can be performed more efficiently by interleaving generation and pruning of the new strategies. First, the algorithm considers a partial mapping \( \beta_t : Obs(N_t) \rightarrow S_{t-1} \) with just one value \( z \in Obs(N_t) \), generates all partial strategies based on this mapping, and prunes dominated strategies. Then it considers a more complex mapping with two values of the observation variable, generates all strategies, and prunes those that are dominated. It continues to generate and prune partial strategies in this way until all values of the observation variable have been considered. This “incremental” approach to generating and pruning observation-based strategies is the idea for which the incremental pruning algorithm is named. It is critical for good performance, and we refer to the literature for further details [10].
4.2.4 Pseudocode and Algorithm Illustration

There are two conditions for removing chance nodes and decision nodes. The conditions for removing chance nodes are the same as traditional Arc Reversal algorithm that is given in Definition 5. However, the conditions for removing decision are relaxed. There is the new condition for removing a decision node.

**Definition 6 (Removable Decision Node)**

> If $D$ is a decision node in an influence diagram with no chance successors. Then $D$ can be removed from the diagram.

Algorithm 11 shows the pseudocode of the Generalized Arc-Reversal algorithm. It takes an influence diagram as input, and then remove a single node in the diagram that satisfied the removable criterion until the only node left is the terminate value node $V_t$. The value of the $V_t$ is the maximum expected utility for the ID.
Algorithm 11: Generalized Arc Reversal Algorithm

**Input:** An Influence Diagram

**Output:** Maximum Expected Utility for the Influence Diagram

1. Add “no-forgetting” arcs
2. Eliminate any barren nodes
3. Initialize strategy node $S$ and strategy value node $V_t$
4. while $\text{pa}(V_t) \neq \emptyset$ do
   5. if there is a removable decision node $D_i$ then
      6. Apply value node removal on all successor value nodes that are not $V_t$
      7. Apply new decision node reduction on $D_i$
      8. Eliminate any resulting barren nodes
   else
      /* there must be a removable chance node $X_i$/
      10. if there is a chance node $X_i$ that all of its successors are value nodes then
          11. Apply value node removal on all successor value nodes that are not $V_t$
          12. if $X_i$ is hidden chance node then
              13. Apply chance node removal on $X_i$
          else if $X_i$ is observed chance node then
              14. Apply new observed node reduction on $X_i$
          else
              15. while the chance node $X_i$ has another chance node $X_j$ do
                  16. Apply Arc reversal between $X_i$ and $X_j$
              end
          17. Apply value node removal on all successor value nodes that are not $V_t$
          18. if $X_i$ is hidden chance node then
              19. Apply chance node removal on $X_i$
          else if $X_i$ is observed chance node then
              20. Apply new observed node reduction on $X_i$
          end
      end
   end
7. return $\{V_t\}$
Algorithm Illustration

In this problem, shown in Figure 4.8, a wildcatter must decide whether or not to drill for oil. The amount of available oil is a hidden state variable with three possible states: dry, wet, and soak. Before deciding whether to drill, the wildcatter can perform a seismic test with three possible outcomes (closed, open, and diffuse), to estimate how much oil is present.

Figure 4.9 and Figure 4.10 show how our algorithm solves this ID using two different orders of reductions. In Figure 4.9, the unobservable chance node for Oil is removed before any other nodes are removed. As a result, there is no need to reason about hidden state or use linear programming to prune dominated strategies. The classic node-removal/arc-reversal algorithm removes nodes in exactly the same order.
Arc-Reversal algorithm performs on oil wildcatter problem with elimination ordering: *Oil, Drill, Seismic, Test*. (A) Initial the strategy node *Strategy* using last decision node *Drill* (B) Apply Arc-Reversal between *Oil* and *Seismic*. (C) Remove *Oil*. (D) Remove *Seismic*. (E) Remove *Test*. 

Figure 4.9
Arc-Reversal algorithm performs on oil wildcatter problem with elimination ordering: *Drill, Seismic, Test, Oil*. (A) Initial the strategy node *Strategy* using last decision node *Drill*. (B) Remove *Seismic*. (C) Remove *Test*. (D) Remove *Oil*.
Figure 4.10 shows an order of reductions that is not possible using the classic approach: the unobservable chance node for Oil is removed after all the other nodes are removed. As a result, removing the observation node (Seismic) and decision node (Test) requires adopting the POMDP approach to reason about hidden state and prune dominated strategies. Our algorithm solves this ID in a small fraction of a second regardless of which of these two orders of reductions is used.

It is interesting to consider the panel shows in Figure 4.10 (C), which shows the ID at the point where all of the nodes have been removed except for the unobservable chance node for Oil (as well as the strategy node and strategy value node). The right part of Figure 4.10 (C) shows the set of undominated strategies at this point, which is represented compactly as an acyclic AND/OR graph we call a strategy graph; OR nodes are squares that represent decisions and AND nodes are circles that represent observations. The solution shown in Figure 4.10 (C) includes an optimal strategy for every possible prior probability distribution for the unobserved state variable for Oil. The decision about which of the four nodes on the left to start with depends on the prior probability distribution. When the prior state probability distribution is 0.5 for dry, 0.3 for wet, and 0.2 for soak, it is best to start in the node marked by an arrow. The final reduction, shows in Figure 4.10 (D), which removes the hidden state variable for Oil, uses this prior probability distribution to select this optimal strategy, and discards the unreachable parts of the strategy graph.
CHAPTER 5
COMPACTED STRATEGY REPRESENTATION

Previous chapters focused on POMDP techniques for solving IDs, one of the key ideas is that by adopting the strategy representation that is used in POMDPs, and use it in solving IDs. However, we did not describe the details of how to represent and construct a strategy for the generalized approaches.

In this chapter, we generalize the concept of a strategy graph that was inspired by the concept of a policy graph as a representation of a policy for a POMDP [27] in a way that better fits IDs, and describe in detail how the generalized variable elimination algorithm for solving IDs can construct a solution that takes the more compact form of a strategy graph [53].

Beside that, we consider a complementary approach to strategy compression that leverages the fact that a strategy graph that contains repeated subgraphs can be represented more compactly – typically, much more compactly – by an equivalent strategy graph. Representing a strategy as a compressed strategy graph has the advantage that it makes the strategy much easier to understand and analyze, which can help make IDs more accessible and useful in practice.
We also propose an approach to approximation that allows further compression of a strategy graph in exchange for bounded-error approximation. Although adoption of this representation of a strategy does not speed up algorithms for solving IDs, it can make the solution constructed by these algorithms easier to interpret and understand.

![Strategy tree for oil wildcatter problem.](image)

**Figure 5.1**

Strategy tree for oil wildcatter problem.

### 5.1 Strategy Graph

Recall that the traditional representation of a strategy is a list of decision rules, \( \Delta = (\delta_1, \ldots, \delta_n) \), one for each decision variable \( D_i \in D \), where each rule is a mapping, \( \delta_i : sp(pa(D_i)) \to sp(D_i) \), that assigns a decision to each instantiation of the parent variables.
Typically, each decision rule is represented by a table, where the dimensionality of the table is equal to the number of parent variables of the corresponding decision node. A table that represents a decision rule is created each time VE eliminates a decision variable. A strategy can also be represented as a strategy tree that is a subtree of a decision tree [36]. Figure 5.1 shows the decision tree for oil wildcatter problem and the part that highlight in blue is the strategy tree for this problem.

In the generalized variable eliminate algorithm, the definition of the strategy is different. Recall that the strategy is defined by equation (3.12), (3.13) and (3.14), which is bottom-up recursively defined, notice that the observed variables are parts of the strategy in the GVE. Therefore, we need a new strategy representation that fit our new strategy definition. We propose to represent a strategy as a graph, called a strategy graph. A strategy tree is a special case of a strategy graph, of course, and so the following definition applies to both strategy trees and strategy graphs. Notice that, the hidden chance variables are never included in the strategy graph.

**Definition 7**

A strategy graph represents a strategy for an influence diagram in the form of a rooted directed acyclic graph with two kinds of nodes:

1. A decision node corresponds to a decision variable of the ID, and has a single outgoing arc labeled by the choice of an action. The outgoing arc leads to a successor node, or to nil if it does not have a successor.
2. An observation node corresponds to an observation variable of the ID, and each of its outgoing arcs is labeled by a non-empty subset of the states of the variable (i.e., the observations), where each state labels at most one arc.

A strategy graph specifies a strategy, as follows. Beginning from the root of the graph, a path from the root to a leaf is followed based on the observed state of each observation node along the path, and the sequence of actions taken is determined by the labels on the outgoing arcs from the decision nodes on the path.

Examples of strategy graphs are shown in Figure 5.14, Figure 5.16 and Figure 5.18. Note that the ordering of variables is the same on every path from the root to a leaf; it is the reverse of the order in which the variables are eliminated in solving the ID. However, it is not necessary for every path from the root to a leaf to include a node for every decision and observation variable of the ID. Note also that every path from the root to a leaf ends at an artificial nil node, which is the only type of node that does not have a successor. The nil node has no function but to serve as a placeholder so that the outgoing edge from the last decision node can have a successor node. It could easily be left out of a display of the strategy graph to improve clarity.

5.2 Constructing a Strategy Graph

Figure 5.2 shows an influence diagram example that we use it to demonstrate how generalized variable elimination algorithm constructs a strategy graph for it. Notice that, we ignore all the conditional probability tables and utility tables since we only want to emphasize how to construct the strategy graph. This influence diagram has $X_1$ and $X_2$ that
are unobserved chance variables, and both $X_1$ and $X_2$ have 2 states; $Z_1$ and $Z_2$ that are observed chance variables, and $Z_1$ has 4 states and $Z_2$ has 2 states; $D_1$ and $D_2$ that are decision variables, and both $D_1$ and $D_2$ have 2 states. We follow the strong elimination ordering $X_2, X_1, D_2, Z_2, D_1, Z_1$ to remove variable one by one using generalized variable elimination algorithm and construct the strategy graph step by step.

At the beginning, we initialize the MEU for this example using Equation 3.8 and we get:

$$MEU = \sum_{Z_1} \left( \max_{D_1} \sum_{Z_2} \max_{D_2} \sum_{X_1} \sum_{X_2} \left( \phi_{X_1}(X_1) \phi_{X_2}(X_2, D_1, X_1) \phi_{Z_2}(Z_2, X_2) \phi_{Z_1}(Z_1, X_1) \right) \times \left( \psi_{V_1}(D_1) + \psi_{V_2}(X_2, D_2) + \psi(S_0) \right) \right)$$
then we remove $X_2$ and $X_1$ using generalized sum marginalization, notice that, since we haven’t eliminated decision variable yet, we only update the potentials but do not construct strategy graph. Remove $X_2$ we get:

$$MEU = \sum_{z_1} \max_{d_1} \sum_{z_2} \max_{d_2} \sum_{x_1} \left( \phi_{X_1}(X_1) \phi_{Z_1}(Z_1, X_1) \phi'_{X_2}(D_1, X_1, Z_2) \right) \times \left( \psi_{V_1}(D_1) + \psi'_{X_2}(D_1, D_2, X_1, Z_2, S_1) \right)$$

where

$$\phi'_{X_2}(D_1, X_1, Z_2) = \sum_{x_2} \left( \phi_{X_2}(X_2, D_1, X_1) \phi_{Z_2}(Z_2, X_2) \right)$$

$$\psi'_{X_2}(D_1, D_2, X_1, Z_2, S_1) = \frac{\sum_{x_2} \left( \phi_{X_2}(X_2, D_1, X_1) \phi_{Z_2}(Z_2, X_2) \left( \psi_{V_2}(X_2, D_2) + \psi(S_0) \right) \right)}{\phi'_{X_2}(D_1, X_1, Z_2)}$$

Remove $X_1$ we get:

$$MEU = \sum_{z_1} \max_{d_1} \sum_{z_2} \max_{d_2} \left( \phi'_{X_1}(Z_1, Z_2, D_1) \right) \times \left( \psi_{V_1}(D_1) + \psi'_{X_1}(Z_1, Z_2, D_1, D_2, S_2) \right)$$

where:

$$\phi'_{X_1}(Z_1, Z_2, D_1) = \sum_{x_1} \left( \phi_{X_1}(X_1) \phi_{Z_1}(Z_1, X_1) \phi'_{X_2}(D_1, X_1, Z_2) \right)$$

$$\psi'_{X_1}(Z_1, Z_2, D_1, D_2, S_2) = \frac{\sum_{x_1} \left( \phi_{X_1}(X_1) \phi_{Z_1}(Z_1, X_1) \phi'_{X_2}(D_1, X_1, Z_2) \psi'_{X_2}(D_1, D_2, X_1, Z_2, S_1) \right)}{\phi'_{X_1}(Z_1, Z_2, D_1)}$$
Then we remove the last decision variable $D_2$, and get:

\[
MEU = \sum_{Z_1} \max_{D_1} \sum_{Z_2} \left( \phi'_{X_1}(Z_1, Z_2, D_1) \right) \times \left( \psi_{V_1}(D_1) + \psi'_{D_2}(Z_1, Z_2, D_1, S_3) \right)
\]

where:

\[
\psi'_{D_2}(Z_1, Z_2, D_1, S_3) = \max_{D_2} \psi_{X_1}(Z_1, Z_2, D_1, D_2, S_2)
\]

Notice that, we are going to construct a strategy graph, shows in Figure 5.3. The dot-lines and dot-shapes are not parts of the strategy graph. However, they are parts of the configurations for the $D_2$, we show them in Figure 5.3 to demonstrate that the strategies are computed by different history configurations. Along with the algorithm keeps running, when all variables are eliminated, some parts of the configurations will become parts of the final strategy graph. Interesting to notice that, although there are only two possible actions for $D_2$, there are 16 strategies in the strategy graph. Obviously, the strategy graph has lots of redundant information, and this is the reason why we need compress the graph. We are going to discuss the strategy compression technique in next section.

![Figure 5.3](image-url)

The strategy graph created after eliminated variable $D_2$. 

115
The strategy graph created after eliminated variable $Z_2$.

Next, we remove observed chance variable $Z_2$, and get:

$$MEU = \sum_{Z_2} \max_{D_1} \left( \phi'_{Z_2}(Z_1, D_1) \right) \times \left( \psi_{V_1}(D_1) + \psi'_{Z_2}(Z_1, D_1, S_4) \right)$$

where:

$$\phi'_{Z_2}(Z_1, D_1) = \sum_{Z_1} \phi'_{X_1}(Z_1, Z_2, D_1)$$

$$\psi'_{Z_2}(Z_1, D_1, S_4) = \frac{\sum_{Z_2} \phi'_{X_1}(Z_1, Z_2, D_1) \psi'_{D_2}(Z_1, Z_2, D_1, S_3)}{\phi'_{Z_2}(Z_1, D_1)}$$

and we continue construct the strategy graph, shows in Figure 5.4.

Next, we remove decision variable $D_1$, and get:

$$MEU = \sum_{Z_1} \phi'_{D_1}(Z_1) \psi'_{D_1}(Z_1, S_5)$$

where

$$\phi'_{D_1}(Z_1) = \max_{D_1} \phi'_{Z_2}(Z_1, D_1)$$

$$\psi'_{D_1}(Z_1, S_5) = \frac{\max_{D_1} \phi'_{Z_2}(Z_1, D_1) \left( \psi_{V_1}(D_1) + \psi'_{Z_2}(Z_1, D_1, S_4) \right)}{\phi'_{D_1}(Z_1)}$$
The strategy graph created after eliminated variable $D_1$. and we continue construct the strategy graph, shows in Figure 5.5.

Finally, we remove the last observed chance variable $Z_1$, and get:

$$MEU = \psi'_{Z_1}(S_6)$$

where:

$$\phi'_{Z_1} = \sum_{Z_1} \phi'_{D_1}(Z_1) = 1$$

$$\psi'_{Z_1}(S_6) = \frac{\sum_{Z_1} \psi'_{D_1}(Z_1) \psi'_{D_1}(Z_1, S_6)}{1}$$

and we continue construct the strategy graph, shows in Figure 5.6.
5.3 Strategy Compression

From Figure 5.6, it is not difficult to see that the strategy graph has lots of duplicate information that can be compressed. As we discussed in the previous chapter, GVE can be considered as a combination of history-based approach and belief-state base approach. The parts in the strategy graph that is constructed by the belief-state base approach do not have duplicate information, it automatically gets compressed. However, the parts in the strategy graph that is constructed different histories do have duplicate information. Actually, we deliberately choose the strong elimination ordering for traditional VE to run GVE on the example we discussed above to show there is redundant information in the strategy graph between different histories.

A strategy graph should not include any redundant information since it does not provide any additional useful information, and we should always remove that redundant information. In order to do so, let us introduce the strategy graph compression technique, the details of the procedure are shown in Algorithm 12.
Algorithm 12: Strategy graph compression.

**Input:** For variable $X_i$ and instantiation $y$ of ancestor variables $Y$, the input is (a) a newly-created node $n$ of a strategy graph, and (b) all the other nodes, $N'$, of the strategy graph.

**Output:** Compressed representation of the strategy graph rooted at node $n$.

1. If $X_i$ is an observation variable:
   
   (a) (Remove zero-probability branches): For the newly-created node $n$, remove an observation $x_i$ and corresponding branch if it has zero probability given instantiation $y$ of $Y$.
   
   (b) (Merge branches with the same predecessor and successor nodes): If two outgoing edges from the newly-created node $n$ have the same successor node, they can be replaced by a single outgoing edge that is labeled by the labels of both original edges, with the interpretation that this edge is followed if either of the conditions corresponding to the labels are true.
   
   (c) (Remove irrelevant observation nodes): If the newly-created node $n$ has only one outgoing edge, then the corresponding observation is irrelevant in this context, and the node can be removed and replaced by a pointer to its successor node.
   
   (d) (Merge isomorphic subgraphs): If the newly-created node $n$ corresponding to observation variable $X_i$ is identical to a node $n'$ already in the strategy graph, in the sense that for every outgoing edge labeled by the same state $x_i$ of $X_i$ for both $n$ and $n'$, the successor node is the same, then nodes $n$ and $n'$ can be merged. The merged node has every outgoing edge and successor node that is part of either $n$ or $n'$.

2. Else if $X_i$ is a decision variable:
   
   (a) (Remove no-op actions): If the newly-created decision node $n$ is a no-op, as could be the case if the utility of every action is worse than the utility of doing nothing, then it can be removed and replaced by a pointer to the successor node of its outgoing edge.
   
   (b) (Merge isomorphic subgraphs): If the newly-created decision node $n$ has the same action for its outgoing edge, and the same successor node, as another node $n'$ in the strategy graph, the two nodes can be merged.
Algorithm 12 uses two kinds of rules to compress a strategy graph. The first kind, consisting of rules 1(a) through 1(c), plus rule 2(a), considers only the newly-added node, its outgoing edges, and their successor nodes in the strategy graph. Note that rules 1(a) through 1(c) are also used by [36] to compress a strategy tree. These rules leverage reachability by removing zero-probability branches; they also leverage context-specific independence [6] to remove nodes that are conditionally irrelevant on one path, but not others. An example of rule 1(b) shows in Figure 5.7 and an example of rule 1(c) shows in Figure 5.8.

The second kind of rule, consisting of rules 1(d) and 2(b), merge isomorphic subgraph; this form of compression is the primary compression source of this technique. These rules consider not only the newly-added node, its outgoing edges, and their successor nodes in the strategy graph; they also consider all of the other nodes of the strategy graph. A newly-created node is merged with an existing node of the strategy graph if they have the same outgoing edges, and the same successor nodes for each edge. Essentially, a new node is merged into an existing node of the strategy graph when the two nodes represent identical
Remove irrelevant observation nodes

subgraphs of the strategy graph. But it is important to note that entire subgraphs do not need to be compared; the subgraphs are identical if the two nodes have the same outgoing edges and successor nodes.

Figure 5.9 illustrates the application of rule 2(d); it shows a newly-added node that is merged with an equivalent node already in the strategy graph. Figure 5.10 illustrates the interaction of this rule with rule 1(a), which prunes zero-probability branches. Because utility is not affected by the successor node of a zero-probability branch, a zero-probability branch can play the role of a “wildcard” that can be matched to anything, allowing additional compression.
On the left is a strategy graph just after node $S_5$ has been added; on the right is the compressed strategy graph.

Nodes $S_4$ and $S_5$ have zero-probability branches that are not shown, allowing them to be merged.
Figure 5.11 shows the strategy graph created by eliminating decision variable $D_2$. Compared to the strategy graph shown in Figure 5.3 that has 16 strategy nodes, the strategy graph in Figure 5.11 only have 2 strategies $S_1$ and $S_2$. Since $D_2$ is the last decision to make, there are only two possible strategies available at this point. The strategy graph after applying the strategy graph compression technique captured that feature.

Figure 5.12 shows the strategy graph after eliminating $Z_2$. Figure 5.13 shows the strategy graph after eliminating $D_1$, notice that, strategy $S5$ and $S6$ in Figure 5.12 are not reachable under the optimal action of $D_1$, therefore, the strategy graph in Figure 5.13 no longer include them. The strategy $S5$ and $S6$ in Figure 5.13 is created by eliminate $D_1$. Figure 5.14 shows the final strategy graph after eliminating $Z_1$. 
Figure 5.13

The strategy graph created by eliminating decision variable $D_1$. 
Figure 5.14

The strategy graph created by eliminating observed chance variable $Z_1$, and it is also the final strategy graph for the demonstrate influence diagram.
It is obvious that the compression rules of Algorithm 12 preserve the equivalence of a strategy. They can also lead to dramatic compression, as the results presented later in the paper show. But the degree of compression is problem-dependent, and often depends on other factors too.

For example, variable ordering matters. If there are several observation variables between one decision variable and the next, then the order in which they are eliminated by the VE algorithm can affect the size of the strategy graph. A similar effect is well-known for ordered binary decision diagrams, where variable ordering can have a significant effect on the the size of a decision diagram, that is, its degree of compression [4].

Elimination-ordering heuristics for VE algorithms typically focus on improving the efficiency with which an ID is solved. But if it is also desirable to minimize the size of the resulting strategy graph, the effect of the elimination order on the size of the strategy graph may also need to be considered. For IDs that are easily solved, it may be useful to solve the ID with several different elimination orders, in the attempt to find one that leads to the smallest strategy graph.

The degree of compression can also depend on tie-breaking issues, although these effects are more subtle. Consider the possibility of adding two different nodes to a strategy graph that have the same utility. Further, suppose that when a choice must be made of which of these nodes to add to the strategy graph, neither one matches a node already in the strategy graph, but one turns out to match a node that is later added to the strategy graph, and one does not. In this case, the tie has to be broken without knowing which way of breaking the tie will lead to greater compression.
Apply Strategy Graph Compression

The GVE algorithm begins to construct the strategy graph when eliminating the last decision variable in the influence diagram. Each newly-created node of the strategy graph has outgoing edges that lead to previously-created nodes of the strategy graph, or else to nil. After a new node is created, the strategy graph compression procedure is invoked. First, we check if we can merge the edges with the same successor strategy nodes. After that, if the strategy graph does not have the identical strategy node to the new strategy node, then add the new strategy node to the strategy graph. Otherwise, do not add it to the strategy graph.

5.4 Bounded-error Approximation

A strategy graph created by the compression procedure of Algorithm 12 may be compressed even further in exchange for bounded-error approximation. We next describe a very simple implementation of this idea.

When eliminating a decision variable $X_i$, consider the possibility of selecting a sub-optimal action for an instantiation $y$ of $Y$, where the sub-optimality is bounded by some threshold $\epsilon > 0$. If the sub-optimal action creates a new node for the strategy graph that can be merged into an existing node, and the optimal action does not, then choosing the sub-optimal action will result in additional compression of the strategy graph, in exchange for bounded-error approximation. The threshold $\epsilon$ could be used to derive a suboptimality bound. However, a much tighter bound can usually be found by simply comparing the
expected utility computed by VE when this approach to approximation is used, and the optimal expected utility.

Unlike other approaches to bounded-error approximation, the motivation for this technique is not to speed up computation, or improve scalability, since it does neither. (Importantly, it does not incur extra overhead either.) Instead, the tradeoff this approach offers between approximation and compression of the strategy graph may be useful as a form of sensitivity analysis, or as an approach to solving IDs with imprecise parameters. For example, it can be used for sensitivity analysis by showing how a strategy can be compressed with limited loss of utility; the part of the strategy graph that is eliminated by simplification can be viewed as the less important part of the strategy graph.

This approach could also be useful for IDs with imprecise parameters. For example, to model such problems, [8] consider interval-valued IDs where the probabilities and utilities of an ID are not represented exactly; instead, they are represented by intervals that bound the uncertainty about the exact values of the parameters. In this framework, the value of an ID cannot be determined exactly; instead, it is represented by an interval. As a result, a VE algorithm for interval-valued IDs may only find a set of potentially optimal strategies, instead of a single strategy, and a secondary criterion may be needed to choose a strategy from this set. Our approach could be used to select the strategy from this set that is most compact, and easiest to understand.
5.5 Experimental Results

We consider some examples that illustrate the benefits of strategy compression technique.

Child diagnosis/treatment problem

Figure 5.15

Influence diagram for diagnosis/treatment problem based on the CHILD belief network, with utility table.

We next consider an ID that is based on a Bayesian network for diagnosing congenital heart disease in a newborn baby with asphyxia [58]. This Bayesian network, named the CHILD network, is well-known in the graphical models community, especially as a test case for learning Bayesian networks. As in the original network, we assume the chance variable for birth asphyxia is always set to true. We transform this Bayesian network into an ID by adding a decision variable that represents a choice of two possible treatments, as
well as a utility node that is a function of both the underlying disease and the treatment. The ID, with utility table, is shown in Figure 5.15. The decision and utility nodes we added are not intended to be medically realistic; they were added simply to create a useful test problem.

We solved this ID using the variable elimination algorithm, modified to construct a strategy graph. Table 5.1 compares the sizes of three different strategy trees, and the equivalent strategy graph. First shown is the original strategy tree before any compression. The “reachable” strategy tree includes only nodes and edges that are reachable by following optimal actions and positive-probability observations (edges). (For this ID, there are no zero-probability observations.) The fully-compressed strategy tree is the result of merging outgoing edges from a node that have the same successor node (rule 1b), and removing observations that are conditionally irrelevant (rule 1c). This compressed strategy tree is the same strategy tree that would be found by the algorithm of [36]. For this ID, compression based on conditional irrelevance dramatically reduces the size of the strategy tree.

<table>
<thead>
<tr>
<th></th>
<th>Strategy Tree</th>
<th>Strategy graph</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>All</td>
<td>Reach.</td>
</tr>
<tr>
<td>CO2Report</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>GruntingReport</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>RUQO2</td>
<td>12</td>
<td>12</td>
</tr>
<tr>
<td>XrayReport</td>
<td>60</td>
<td>60</td>
</tr>
<tr>
<td>LowerBodyO2</td>
<td>180</td>
<td>180</td>
</tr>
<tr>
<td>LVH_Report</td>
<td>360</td>
<td>360</td>
</tr>
<tr>
<td>Age</td>
<td>1,080</td>
<td>1,080</td>
</tr>
<tr>
<td>Treatment</td>
<td>2,160</td>
<td>1,080</td>
</tr>
<tr>
<td>Total</td>
<td>3,858</td>
<td>2,778</td>
</tr>
</tbody>
</table>

Table 5.1: Comparison of the number of nodes in the strategy trees, and equivalent strategy graph, for the CHILD influence diagram.
The column *All* shows the number of nodes in the strategy tree before any compression; the column *Reach.* shows the number of reachable nodes under an optimal strategy; the column *Compr.* shows the number of nodes in a fully-compressed strategy tree (with many irrelevant nodes removed based on context-specific independence); and the last column shows the number of nodes in the strategy graph.

The last column of Table 5.1 shows the size of the strategy graph after isomorphic subgraphs are merged. The results show how more compression is achieved by transforming a strategy tree into an equivalent strategy graph. The optimal strategy graph is shown in Figure 5.16, and has 17 nodes and 27 edges. Note that Table 5.1 shows the number of nodes in each level of the tree/graph, as well as the total number of nodes.
Figure 5.16

Optimal strategy graph for the Child influence diagram.
Arthronet problem

The last ID we consider represents a realistic model of medical decision making for total knee arthroplasty [34], the influence diagram of this problem shows in Figure 5.17. Named Arthronet, it has eleven chance variables (ten of them are observable), four decision nodes, and four utility nodes. The ID and its parameters are available in the software package OpenMarkov.¹

One reason for using this ID as a test problem is that [36] report that the strategy tree for this ID found by their algorithm has too many nodes to be easily understood by a user; our results show that even a compressed strategy tree has 1,747 nodes. By contrast, our

¹www.probmodelxml.org/networks/
algorithm finds a strategy graph that is more than an order of magnitude smaller, with only 143 nodes. These compression results are shown in Table 5.2.

In fact, the strategy graph can be compressed even further using bounded-error approximation. Figure 5.18 shows the strategy graph that results from using a suboptimality bound of 0.2 when choosing an action for a decision node; in this approach, the bounded-suboptimal action is chosen that leads to the most compression. The resulting strategy graph has only 76 nodes, and yet its utility is 2.05655, which is very close to the optimal utility of 2.05668.
<table>
<thead>
<tr>
<th>Variable</th>
<th>Decision Tree</th>
<th>Strategy tree Reach.</th>
<th>Strategy tree Comp.</th>
<th>Strategy graph</th>
<th>Approx. S. G.</th>
</tr>
</thead>
<tbody>
<tr>
<td>BMI</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Diabetes</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>Antibiotic allergy</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>Implant prosthesis</td>
<td>16</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>7</td>
</tr>
<tr>
<td>Knee motion</td>
<td>48</td>
<td>16</td>
<td>16</td>
<td>8</td>
<td>7</td>
</tr>
<tr>
<td>Ischemia</td>
<td>144</td>
<td>32</td>
<td>32</td>
<td>16</td>
<td>10</td>
</tr>
<tr>
<td>Drained CC</td>
<td>288</td>
<td>64</td>
<td>64</td>
<td>32</td>
<td>14</td>
</tr>
<tr>
<td>C reactive protein</td>
<td>576</td>
<td>128</td>
<td>115</td>
<td>37</td>
<td>12</td>
</tr>
<tr>
<td>ESR</td>
<td>1,152</td>
<td>256</td>
<td>146</td>
<td>19</td>
<td>6</td>
</tr>
<tr>
<td>Make scintigraphy</td>
<td>2,304</td>
<td>256</td>
<td>146</td>
<td>8</td>
<td>5</td>
</tr>
<tr>
<td>Seq. Ga67 Tc99</td>
<td>6,912</td>
<td>430</td>
<td>201</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>Synovial biopsy</td>
<td>13,824</td>
<td>430</td>
<td>201</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>Froz. section PMN</td>
<td>41,472</td>
<td>514</td>
<td>402</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Remove prosthesis</td>
<td>82,944</td>
<td>514</td>
<td>402</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td><strong>Total nodes</strong></td>
<td><strong>149,694</strong></td>
<td><strong>2,662</strong></td>
<td><strong>1747</strong></td>
<td><strong>143</strong></td>
<td><strong>76</strong></td>
</tr>
</tbody>
</table>

Table 5.2: Comparison of number of nodes in the strategy trees, and equivalent strategy graph, for the Arthronet influence diagram.

The column *All* shows the number of nodes in the strategy tree before any compression; the column *Reach.* shows the number of reachable nodes under an optimal strategy, with zero-probability branches also removed; the column *Comp.* shows the number of nodes in a fully-compressed strategy tree; and the last column shows the number of nodes in the strategy graph.
Figure 5.18

Bounded-suboptimal strategy graph for Arthronet influence diagram.
CHAPTER 6

CONCLUSIONS

6.1 Contributions

We have developed two algorithms that are: *generalized arc-reversal algorithm* and *generalized variable elimination algorithm* for influence diagram evaluation that includes operations that allow an ID to be solved by a generalization of the dynamic programming approach to solving POMDPs. By using POMDP techniques to reason about hidden state, both of the new algorithms can postpone the elimination of unobserved chance variables until decision and observation variables are eliminated. Therefore, both of the algorithms allow a more flexible elimination ordering, which can improve the algorithms scalability in many cases, especially for solving complex, multi-stage problems. On the other hand, the two new algorithms also allow variables to be removed in the same order as they are removed by the traditional ID algorithms, which ensures that they can perform at least as well as the traditional algorithms.

We proposed a heuristic for selecting the variable elimination ordering that could lead a better performance for generalized variable elimination algorithm since generalized variable elimination algorithm has a more relaxed constraint on the variable elimination ordering. We have also introduced an approach to representing the solution for influence diagrams as an easier-to-understand strategy graph. This approach is in keeping with the
original motivation for influence diagrams, which is to facilitate understanding and communication with users. We have also shown that a strategy graph can be further compressed in exchange for bounded-error approximation, making it possible to perform a sensitivity analysis that tests which parts of a strategy can be omitted without significantly affecting performance, as well as simplifying the strategy further.

6.2 Future Research Directions

We have sketched the basic idea of integrating POMDPs’ approach to IDs’ approach for solving IDs in this dissertation. There are several potential directions to explore in future work, including development of more sophisticated heuristics for variable elimination ordering, development of approaches to approximation that leverage this more general perspective about the relationship between POMDPs to IDs. Beside on that, the idea of combine history-based dynamic programming with belief-based dynamic programming can be expanded to other graphical models such as interactive dynamic influence diagrams [61].
REFERENCES


