Value-Based Abstraction Functions for Abstraction Sampling (Extended Background)

Bobak Pezeshki[1] **Kalev Kask**[1] **Alexander Ihler**[1] **Rina Dechter**[1]

[1]University of California, Irvine

Abstract

For revised supplemental materials, please visit https://ics.uci.edu/~dechter/publications.html. In addition to providing a glossary of terms, abbreviations, and notation, this document aims to provide readers with background on topics that are foundational to the concepts that are discussed in the main paper. The most up-to-date version of this document - as well as other supplemental materials - can be found on the Dechter Lab publications page.

CONTENTS

i. Glossary **3**

ii. Abbreviations **5**

1 Background: Graphical Models **6**

1.1 Discrete Graphical Models . . . 6

1.2 Graphical Model Notation . . . 6

1.3 Primal Graph . . . 6

1.4 Simple Example . . . 7

1.5 Pseudo Tree . . . 8

1.6 Some Well-Known and Important Graphical Model Tasks . . . 9

1.6.1 Task Formalizations . . . 9

1.6.2 Difficulty . . . 9

2 Background: Fundamental Schemes **10**

2.1 Search . . . 10

2.1.1 OR Search Spaces . . . 10

2.1.2 AND/OR Search Space . . . 11

2.1.3 Search Space Notation . . . 13

2.1.4 Important Quantities in AND/OR Search Spaces 13
2.2 Inference 15
2.2.1 Variable Elimination 15
2.2.2 Bucket Elimination 16
2.2.3 Induced Width (w*) 17

GLOSSARY

bucket

Data structure used in **bucket elimination**, each corresponding to a particular variable to be eliminated, that collects functions to be processed during an elimination step. Processing of a bucket involves applying an elimination operator (such as maximization or marginalization) over the bucket's variable to the combined functions of the bucket. 16

bucket tree

In the context of **bucket elimination**: A directed tree with nodes corresponding to buckets and directed edges corresponding to the origin and destination of computed messages. 16

bucket message

The function resulting after processing of a bucket during a **bucket elimination** elimination step. 16

bucket elimination

A variable elimination framework that uses a dynamic programming approach leveraging commutability of mathematical expressions. 3, 16

configuration

A joint assignment to a set of variables. (See **full configuration** and **partial configuration** for more specific uses of the term). 6

constrained ordering

An elmination ordering that satisfies given constraints for some variables to be eliminated before others. 9

elimination order

Order in which to process and eliminate variables during variable elimination inference. 16

full configuration

A joint assignment to all the variables of a graphical model. 3, 6, 10

graphical model

($\mathcal{M} = \langle \boldsymbol{X}, \boldsymbol{D}, \boldsymbol{F} \rangle$). Mathematical tool for modeling complex systems composed of a set of variables $\boldsymbol{X}$, a set of domains $\boldsymbol{D} = \{D_X | X \in \boldsymbol{X}\}$ for each variable X, and a set of functions $\boldsymbol{F}$ with each function defined over a subset of the model's variables $\alpha \subseteq \boldsymbol{X}$. 6

induced width

(w^*) One less than the largest clique size of the **induced primal graph**. 17

induced primal graph

With respect to a variable elimination procedure: the primal graph of a graphical model augmented with additional edges corresponding to the scope of messages created through the variable elimination procedure. 3

marginal maximum a-posteriori

(MMAP). The marginal likelihood associated with the configuration of a target subset of variables $\boldsymbol{Q}$ that maximizes their marginal likelihood.

In the context of discrete graphical models without evidence, with $\boldsymbol{Q} \subset \boldsymbol{X}$, $\boldsymbol{S} = \boldsymbol{X} \setminus \boldsymbol{Q}$ be the variables to sum over, $\boldsymbol{F_Q} = \{f_\alpha \mid \alpha \subseteq \boldsymbol{Q}\}$ be the set of functions defined only over $\alpha \in \boldsymbol{Q}$, and $\boldsymbol{F_S} = \boldsymbol{F} \setminus \boldsymbol{F_Q}$ be functions that include some $X \in \boldsymbol{S}$ in their scope,

$$MMAP = \max_{\boldsymbol{Q}} \sum_{\boldsymbol{S}} \prod_{f_\alpha \in \boldsymbol{F}} f_\alpha(\boldsymbol{q} \cup \boldsymbol{s}) \tag{1}$$

$$= \max_{\boldsymbol{Q}} \prod_{f'_\alpha \in \boldsymbol{F_S}} f'_\alpha(\boldsymbol{q}) \sum_{\boldsymbol{S}} \prod_{f''_\alpha \in \boldsymbol{F_S}} f''_\alpha(\boldsymbol{q} \cup \boldsymbol{s}) \tag{2}$$

5, 6, 9

maximum a-posteriori

(MAP). With respect to a graphical model, the likelihood value associated with the **most probable explanation** or **MPE**.
In the context of discrete graphical models without evidence, $MAP = \max_{\boldsymbol{Q}=\boldsymbol{X}} \prod_{f_\alpha \in \boldsymbol{F}} f_\alpha(\boldsymbol{q})$. 5, 6, 9

most probable explanation

(MPE). Assignment to variables the variables of a graphical model that maximizes the conditional probability of the observed evidence.
In the context of discrete graphical models without any evidence, $MPE = \mathrm{argmax}_{\boldsymbol{Q}=\boldsymbol{X}} \prod_{f_\alpha \in \boldsymbol{F}} f_\alpha(\boldsymbol{q})$. 4, 5, 6, 9

partial configuration

A joint assignment to a subset of the variables of a graphical model. 3, 6

partition function

(Z). A mathematical quantity that characterizes the distribution among a system's possible states and serves as a normalizing constant for calculating probabilistic measures associated with these states.
In the context of discrete graphical models, $Z = \sum_{\boldsymbol{X}} \prod_{f_\alpha \in \boldsymbol{F}} f_\alpha(\boldsymbol{x})$. 6, 9, 16

primal graph

The primal graph of a graphical model captures an underlying structure of the model, where each node corresponds to a random variable of the model and edges connect any two variables that are both part of a function's scope thus indicating direct dependencies between variables. 6

scope

The set of variables a function is defined over. Denoted as α in the general case. 6

variable elimination

(VE). An inference technique that involves an ordered computational processing of variables, each elimination step removing the processed variable from the resulting expression. 15

ABBREVIATIONS

MAP
: **Maximum a-posteriori**. 9, 16

MMAP
: **Marginal maximum a-posteriori**. 9

MPE
: **Most probable explanation**. 4, 9

1 BACKGROUND: GRAPHICAL MODELS

Graphical models, such as a Bayesian or a Markov networks Pearl [1988], Darwiche [2009], Dechter [2013], are mathematical tools for modeling complex systems, each composed of a set of variables with defined domains and functions defined over subsets of the variables. The functions capture local dependencies of the subset of variables they are defined on, those variables known as the function's **scope**. The functions of a graphical model often represent a factorization of a global function over all the variables. An assignment to all of the variables (referred to as a **full configuration**) represents a possible state of the modeled system.

Graphical models are constructed not only to model a system, but also to provide a means of efficiently answering specific queries of interest via exploitation of the model's structure. Some common computational tasks are

- determination of the **partition function**: a normalization constant necessary for computing probabilistic quantities
- determination of the **MPE** (**most probable explanation**): the most probable full configuration given a **partial configuration** (assignments to a subset of the variables) known as observation or evidence. Additionally, the associated likelihood corresponding to the MPE, known as the **MAP** (**maximum a-posteriori**) value, and can also be queried in kind
- determination of the **MMAP** (**marginal maximum a-posteriori**) configuration: the configuration of a target subset of variables that maximizes their marginal likelihood

(More details about common graphical model queries to be provided in Section 1.6: Some Well-Known and Important Graphical Model Tasks).

1.1 DISCRETE GRAPHICAL MODELS

Considering the discrete space, a discrete graphical model can be defined as a 3-tuple $\mathcal{M} = \langle \boldsymbol{X}, \boldsymbol{D}, \boldsymbol{F} \rangle$, where

- $\mathbf{X}$ is a set of variables for which the model is defined over
- $\mathbf{D} = \{D_X : X \in \mathbf{X}\}$ is a set of finite domains, one for each $X \in \mathbf{X}$, defining the possible values each X can be assigned
- Each $f_\alpha \in \mathbf{F}$ (sometimes denoted $f \in \mathbf{F}$ for simplicity) is a real-valued function defined over a subset of the model's variables $\alpha \subseteq \mathbf{X}$, known as the function's **scope**, for which the function defines local interactions. More concretely, if we let D_α denote the Cartesian product of the domains of the variables in α, then $f_\alpha : D_\alpha \rightarrow \mathbb{R}_{\geq 0}$. These functions can be expressed as tables for which there is a real valued output associated with every possible input $d_\alpha \in D_\alpha$ (ie. every possible joint assignment - or **configuration** - to all of the variables in α).

1.2 GRAPHICAL MODEL NOTATION

Capital letters (X) represent variables and small letters (x) represent their values. Boldfaced capital letters ($\mathbf{X}$) denote a collection of variables, $|\mathbf{X}|$ its cardinality, $D_{\boldsymbol{X}}$ their joint domains, and $\boldsymbol{x}$ a particular realization in that joint domain. Abusing notation, operations denoted $\bigoplus_{\boldsymbol{X}}$ (ex. $\sum_{\boldsymbol{X}}$) imply...

$$\begin{aligned} \bigoplus_{\boldsymbol{X}} &\iff \bigoplus_{\boldsymbol{x} \in D_{\boldsymbol{X}}} \\ &\iff \bigoplus_{x_1 \in D_{X_1}} \bigoplus_{x_2 \in D_{X_2}} \cdots \bigoplus_{x_{|\boldsymbol{X}|} \in D_{X_{|\boldsymbol{X}|}}} \end{aligned} \tag{3}$$

Furthermore, given a function f_α with scope α, a super-set of variables β s.t. $\alpha \subseteq \beta$, a particular configuration $\mathbf{b}$ of β, and $\mathbf{a} := \{X \leftarrow x | X \leftarrow x \in \mathbf{b} \text{ and } X \in \alpha\}$ (ie. the subset of assignments in $\mathbf{b}$ corresponding to variables in α),

$$f_\alpha(\mathbf{b}) \implies f_\alpha(\mathbf{a}) \tag{4}$$

1.3 PRIMAL GRAPH

A **primal graph** $\mathcal{G} = \langle \mathbf{X}, \mathbf{E} \rangle$ of a graphical model $\mathcal{M}$ associates each variable of $\mathcal{M}$ with a corresponding node in a one-to-one fashion such that arcs $e \in \mathbf{E}$ connect nodes whose variables appear in the scope of the same local function. To simplify, we abuse notation by using the same symbols to refer to primal graph nodes as their corresponding variables in $\mathcal{M}$.

(For those familiar, note that the primal graph corresponds to a Markov Random Field graph representation of the model). The primal graph is a useful tool for graphical model algorithms' exploitation of the model's local structure.

1.4 SIMPLE EXAMPLE

Consider a simple model that relates temperature and humidity to the chance of rain, and temperature and elevation to the chance of different oxygen levels. Let us choose binary variables $\mathbf{X} = \{T, H, R, E, O\}$ to represent these different levels and construct a corresponding graphical model $\mathcal{M} = \langle \boldsymbol{X}, \boldsymbol{D}, \boldsymbol{F} \rangle$ where

- T has corresponding domain $D_T = \{low, high\}$ representing high and low temperature
- H has corresponding domain $D_H = \{low, high\}$ representing high and low humidity
- R has corresponding domain $D_R = \{no, yes\}$ representing the presence or absence of rain
- E has corresponding domain $D_E = \{low, high\}$ representing the high or low elevation
- O has corresponding domain $D_O = \{low, high\}$ representing the high or low oxygen levels

and having five functions

- $f_T(T)$ representing the marginal probability of the temperature being low or high, $p(T)$
- $f_H(H)$ representing the marginal probability of the humidity being low or high, $p(H)$
- $f_E(E)$ representing the marginal probability of the elevation being low or high, $p(E)$
- $f_{T,H,R}(T, H, R)$ representing the conditional probability of rain given levels of humidity and temperature, $p(R \mid T, H)$
- $f_{T,E,O}(T, E, O)$ representing the conditional probability of high vs. low oxygen concentrations given the temperature and elevation levels, $p(O \mid T, E)$

defined by the following tables, respectively:

T	$p(T)$
low	0.60
$high$	0.40

H	$p(H)$
low	0.75
$high$	0.25

E	$p(E)$
low	0.80
$high$	0.20

T	H	R	$p(R \mid T, H)$
low	low	no	0.90
low	low	yes	0.10
low	$high$	no	0.20
low	$high$	yes	0.80
$high$	low	no	0.95
$high$	low	yes	0.05
$high$	$high$	no	0.60
$high$	$high$	yes	0.40

T	E	O	$p(O \mid T, E)$
low	low	low	0.30
low	low	$high$	0.70
low	$high$	low	0.75
low	$high$	$high$	0.25
$high$	low	low	0.60
$high$	low	$high$	0.40
$high$	$high$	low	0.80
$high$	$high$	$high$	0.20

and where we make independence assumptions allowing the joint distribution $P(T, H, R, E, O)$ to factorize to the probability functions represented by the model such that:

$$\begin{aligned} p(T, H, E, R, O) &= p(T) \cdot p(H \mid T) \cdot p(E \mid T, H) \cdot p(R \mid T, H, E) \cdot p(O \mid T, H, E, R) \\ &= p(T) \cdot p(H) \cdot p(E) \cdot p(R \mid T, H) \cdot p(O \mid T, E) \\ &= f_T(T) \cdot f_H(H) \cdot f_E(E) \cdot f_{T,H,R}(T, H, R) \cdot f_{T,E,O}(T, E, O) \\ &= \prod_{f_\alpha \in \mathbf{F}} f_\alpha(\alpha) \end{aligned} \tag{5}$$

You can see that the primal graph consists of nodes representing T, H, R, E, and O and has edges between each pair of $\{T, H, R\}$ since each pair appears together in at least one $f_\alpha \in \mathbf{F}$, and similarly between each pair of $\{T, E, O\}$.

With the model and primal graph defined, we can then use a variety of algorithms over graphical models to efficiently answer queries about the model. One such query could be to find the probability corresponding to the mode of our modeled distribution - namely to find the probability associated with the most likely full configuration. Throughout the next several

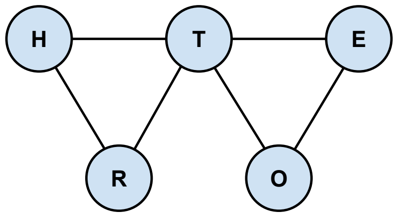


Figure 1: Primal graph of the example model described above.

sections, we will describe various queries and computational schemes commonly used with graphical models starting next by describing the general framework of variable elimination, which we will use to show one way to compute our example query.

1.5 PSEUDO TREE

Given a variable ordering, directed tree called a ***pseudo tree*** $\mathcal{T} = (\mathbf{X}, \mathbf{E})$ can be constructed relative to a graphical model $\mathcal{M}$. Each node of the pseudo tree corresponds one-to-one with a node in $\mathcal{M}$. As before, to simplify, we abuse notation by using the same symbols to refer to pseudo tree nodes as their corresponding variables in $\mathcal{M}$. The tree will be structured such that the nodes follow the provided variable ordering - namely, each node is a descendant of only nodes that come before it in the provided ordering - and such that any branching in the pseudo tree corresponds to existing conditional dependencies in $\mathcal{M}$ - namely, sibling branches are conditionally independent of each other given assignments to their ancestor variables in the pseudo tree.

A pseudo tree $\mathcal{T}$ can be constructed by the following steps:

1. create a dummy root X_0
2. add the first variable in the ordering $X_{i=1}$ as the child of the dummy root
3. for the next variable in the ordering X_{i+1}:
   i. choose an existing variable X_p in the partially constructed pseudo tree $\mathcal{T}'$ such that, given assignments to X_p and its ancestors, X_{i+1} is conditionally independent of all existing descendants of X_p and of existing descendants of its ancestors.
   ii. add X_{i+1} to $\mathcal{T}'$ as the child of X_p
4. repeat step 3 until all variables in the ordering (and thus $\mathcal{M}$) have been added to the tree

Note that edges in the primal graph of the model either exist in the pseudo tree as directed edges or would exist as back arcs, but never cross arcs.

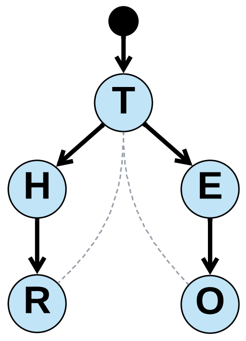


Figure 2: An example pseudo tree for the model described in Section 1.4: Simple Example based on ordering T, H, R, E, O. Here the dummy root node is explicitly shown, however it is typically hidden for simplicity.

Increasing Pseudo Tree Branching. In order to capture the maximal number of conditional independences given an ordering, step 3.i. can be altered to choose the earliest variable in the ordering that satisfies the said condition, thus leading to earlier branching in the tree.

Pseudo Tree Uses. One use of the pseudo tree is as a schematic of bucket elimination. Mores specifically, message passing from [mini] bucket elimination with an elimination ordering that is the reverse of o will follow a path from the leaves to the root of $\mathcal{T}$. (TODO: EXAMPLE). Furthermore, the pseudo tree can act as a blue print for constructing search space graphs of $\mathcal{M}$ as will be described in the next section. In combination, given an ordering o and corresponding search space and bucket elimination, the messages from the bucket elimination can act as heuristics guiding the search at each level.

1.6 SOME WELL-KNOWN AND IMPORTANT GRAPHICAL MODEL TASKS

There are a plethora of queries that a graphical model can lend itself to answering. Here we will describe four traditionally important tasks in particular: determination of the

- **Partition function (Z)**
- **Maximum a-posteriori (MAP)**
- **Most probable explanation (MPE)**
- **Marginal maximum a-posteriori (MMAP)**

1.6.1 Task Formalizations

Definition 1.6.1.1 below provides the formalization of these tasks respectively.

Definition 1.6.1.1 (Z, MAP, MPE, and MMAP)**.** *Given a graphical model* $\mathcal{M} = (\boldsymbol{X}, \boldsymbol{D}, \boldsymbol{F})$,

$$Z = \sum_{\boldsymbol{X}} \prod_{\boldsymbol{F}} f(\boldsymbol{x}); \tag{6}$$

$$MAP = \max_{\boldsymbol{X}} \prod_{\boldsymbol{F}} f(\boldsymbol{x}); \tag{7}$$

$$MPE = \operatorname*{argmax}_{\boldsymbol{X}} \prod_{\boldsymbol{F}} f(\boldsymbol{x}); \tag{8}$$

$$MMAP = \max_{\boldsymbol{Q} \subset \boldsymbol{X}} \sum_{\boldsymbol{S} = \boldsymbol{X} \setminus \boldsymbol{Q}} \prod_{\boldsymbol{F}} f(\boldsymbol{q} \cup \boldsymbol{s}) \tag{9}$$

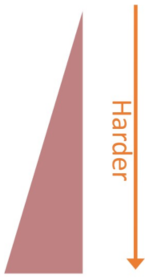
The **partition function**, **Z**, is mathematical quantity that characterizes the distribution among a system's possible states. It is often used as a normalization constant for computing probabilities. **MPE** is a full configuration that maximizes the value of the model defined as the product of all of its functions, and **MAP** outputs that value. From a probabilistic model standpoint, this corresponds to finding the assignment to the variables that are most likely under the model, and the corresponding likelihood value, respectively. Given evidence (ie. a given assignment to a subset of the variables), the MPE (constrained to be consistent with the evidence) corresponds to finding the assignment to the rest of the variables that makes the evidence most likely to occur (thus the name "most probable explanation"). **MMAP** is similar to MAP with the exception that the model value is now defined with respect to the marginalization of a subset of the variables denoted as the "sum" or $\boldsymbol{S}$ variables, and so the maximization is with respect to the remaining set denoted as the "query" or $\boldsymbol{Q}$ variables. Although not commonly referred to, and thus omitted here, there can also be a corresponding MMPE task.

1.6.2 Difficulty

Summation tasks such as computing the partition function require consideration of the entire state-space (exponential in the number of variables) to compute accurately and are generally #P-hard. Since summation operations commute freely, when variable elimination algorithms are used for these tasks (with the variable ordering corresponding to the order in which variables are summed over in Equation 6), they can be used with any variable ordering. Pure homogeneous optimization such as MAP and MPE inference, whose solutions can be confirmed in polynomial time, are easier but still NP-Hard to compute. Since optimization operators can freely commute with others of their own kind (ex. max operators can commute with each other, or min operators can commute with each other), variable elimination for these tasks can also use of any variable ordering. Mixed inference tasks, however, are often more difficult to compute as they involve operators that do not commute. In the example of MMAP (Equation 9), the summation operations must be computed before maximization, and thus restricts the variable orderings that can be used. In practice, this **constrained ordering** can lead to inference over graphs of much greater widths (see Section 2.2.3: Induced Width (w*) for more details) and thus are more difficult to compute.

This hierarchy of difficulties is summarized in Figure 3.

▸ Max-Inference	$f(\mathbf{x}^*) = \max_{\mathbf{x}} \prod_{\alpha} f_\alpha(\mathbf{x}_\alpha)$
▸ Sum-Inference	$Z = \sum_{\mathbf{x}} \prod_{\alpha} f_\alpha(\mathbf{x}_\alpha)$
▸ Mixed-Inference	$f(\mathbf{x}_M^*) = \max_{\mathbf{x}_M} \sum_{\mathbf{x}_S} \prod_{\alpha} f_\alpha(\mathbf{x}_\alpha)$



- **NP-hard**: exponentially many terms

Figure 3: Hierarchy of difficulties for three classes of graphical model inference tasks.

2 BACKGROUND: FUNDAMENTAL SCHEMES

In order to be able to solve important and difficult tasks in discrete graphical model framework, efficient algorithms are necessary. Several foundational schemes serve as the backbone of a myriad of graphical model tasks. Next we outline these schemes and their properties.

2.1 SEARCH

We will start by describing fundamental search schemes used for solving tasks formulated as probabilistic graphical models.

2.1.1 OR Search Spaces

A graphical model can be cast into a search space in order to explore different configurations of the model. Figure 4 shows a classical search space (also known as an "OR" search space) of the model described in Section 1.4: Simple Example adhering to a search order that explores possible assignments to variable T, then H, then R, then E, then O. (For simplicity, we abbreviate domain values of low or no instead with the value 0, and $high$ or yes with 1).

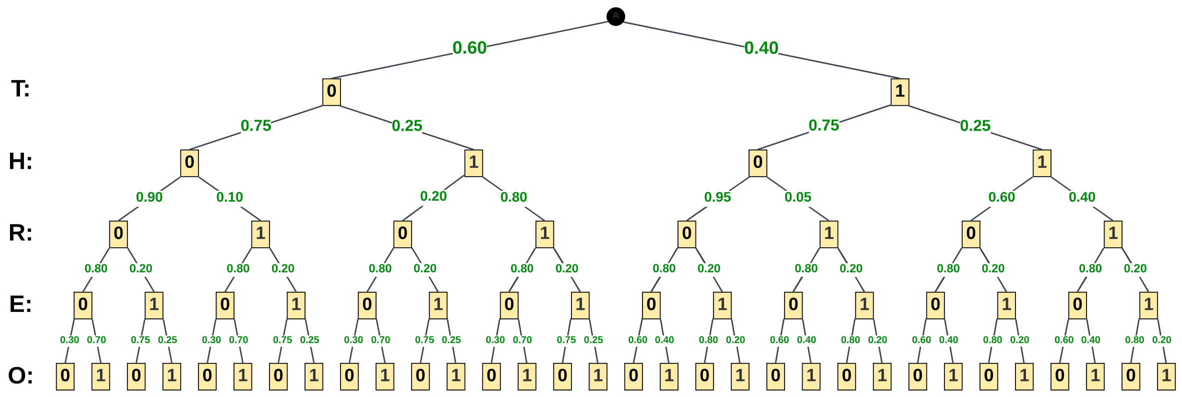


Figure 4: Example classical OR search space for the discrete graphical model described in Section 1.4: Simple Example adhering to a search order that explores possible assignments to variable T, then H, then R, then E, then O. For simplicity, we abbreviate domain values of low or no instead with the value [0], and $high$ or yes with [1].

As we follow a path down the tree, each successive level corresponds to an assignment to the next variable in the ordering. Thus a path from the dummy root to a leaf corresponds to a **full configuration**. Given the search tree was built for our model that is meant to capture a factorized global function (in this case a factorized probability distribution), the search tree is constructed so the arc into a node n associated with variable X has a cost $c(n)$ equal to the product of functions $f_\alpha \in \boldsymbol{F}$ such that the path to n_X fully instantiates all $X' \in \alpha$ and such that $X \in \alpha$ [Dechter and Mateescu, 2007]. In other words,

$c(n)$ equal to the product of functions $f_\alpha \in \boldsymbol{F}$ such the variable represented by n is in f's scope and that the path to n captures an assignment to every other variable in its scope. (If no such functions exist, the arc is vacuously assigned a value of 1.00.

2.1.2 AND/OR Search Space

Often, assignments to earlier variables in the search ordering results in conditional independences between sub trees of later layers. For example, given our model from Section 1.4: Simple Example conditioning on variable T (ie. giving an assignment to T) causes E and O to become independent of H and R. In the OR search space we can see this phenomenon by noticing that the edge cost into and sub tree under nodes of E under the same assignment of T but different assignments to H and R are duplicates. (Figure 5 shows this more explicitly).

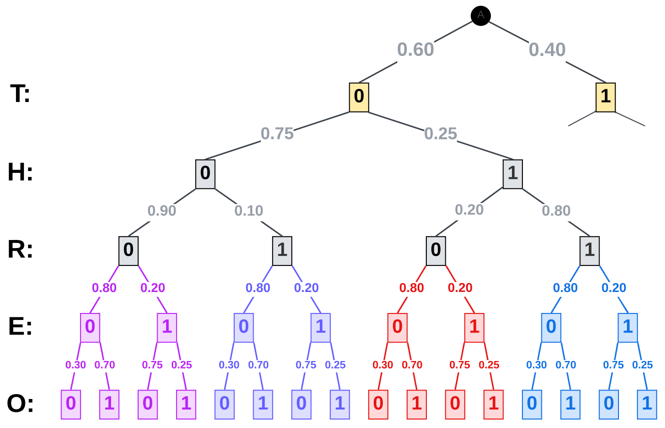


Figure 5: Conditional independence of E and O from H and R given assignment $T = 0$ is shown in the search space from Figure 4. Notice that each distinct assignment to H and R leads to equivalent sub trees of E and O (each highlighted in a different colors for easier comparison).

We can take advantage of such conditional independences to construct a more compact search space that we will call an AND/OR search space. Since such conditional independences are inherently captured by pseudo trees (Section 1.5: Pseudo Tree), we can use pseudo trees to guide the construction of AND/OR search spaces.

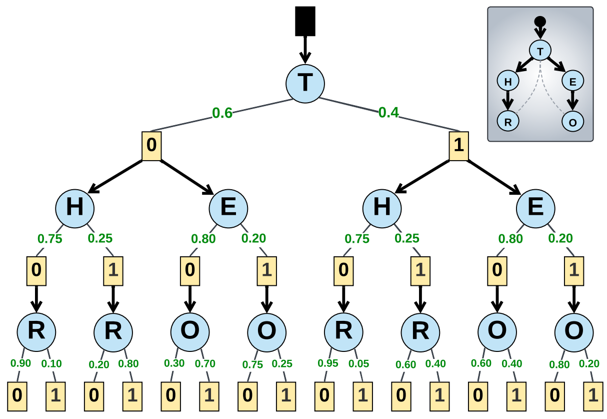
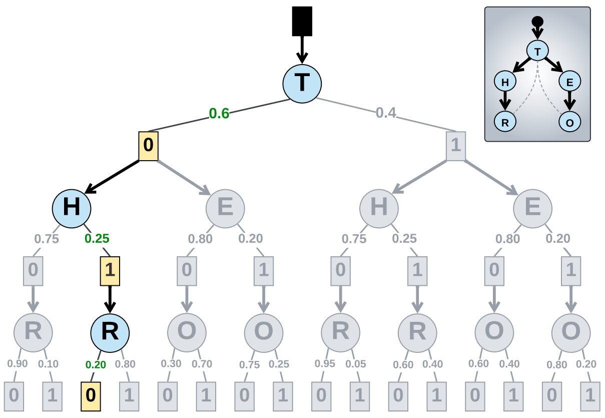


Figure 6: Example of a more compact AND/OR search space for the discrete graphical model described in Section 1.4: Simple Example guided by the pseudo tree from Section 1.5: Pseudo Tree. For simplicity, we abbreviate domain values of low or no instead with the value $[0]$, and $high$ or yes with $[1]$.


(a) Paths from root to leaves in AND/OR search spaces do not necessarily correspond to full configurations. In the example shown here, the highlighted path captures a partial configuration with assignments $T = 0, H = 1, R = 0$, but omits assignments to E and O.

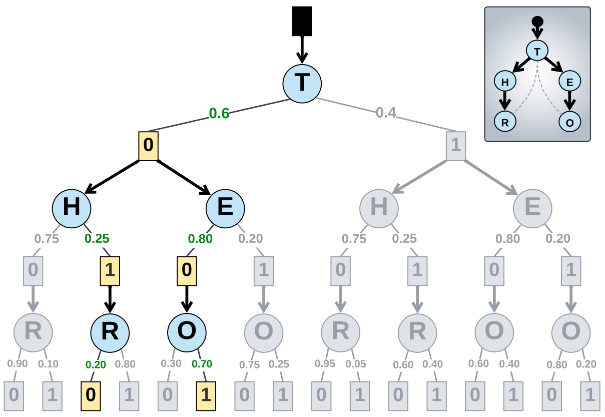

(b) To capture a full configuration in an AND/OR search space, we must capture *all* variables that branch from paths extended leading to a subtree of the full search space that includes all variables of the model.

Figure 7: Figure (a) shows an example AND/OR search space where a path corresponds only to a partial configuration of the variables in the model. Figure (b) shows how a full configuration can be captured.

Figure 6 shows the AND/OR search space that results from using the pseudo tree from figure Figure 2 as a guide. Within AND/OR search spaces, nodes corresponding to assignments to variables (these are AND nodes; the yellow rectangles) are directly associated with a parent node corresponding to their respective variable (OR nodes; the blue circles). Branching in the guiding pseudo tree capturing conditional independences are also seen as branching in the AND/OR search space. In the example provided, we see a branching under T in the guiding pseudo tree that captures the conditional independence of H and R from E and O given assignment to T. In the corresponding AND/OR search space, under each assignment of T (namely under each AND child node of T) we see a branching leading to distinct sub trees - one for H and R, and one for E and O. Through capturing such decomposition, the search space can be greatly compacted.

It is important to note that a path from root to leaf in an AND/OR search space does not necessarily capture a full configuration. For example, the path from root to leaf highlighted in Figure 7a captures a partial configuration corresponding only to assignments $T = 0, H = 1, R = 0$, omitting assignments to E and O.

In contrast, note that a full configuration (such as the one captured in Figure 7b for $T = 0, H = 1, R = 0, E = 0, O = 1$) consists of a *sub tree* such that at any point following a path from the root towards the leaves that a variable branching occurs under an AND node, *all* children OR nodes are included in the final subtree. The cost of a full configuration in an AND/OR subtree can be computed by applying a related combination operation (often multiplication) to the cost of each arc traversed.

Finally, we can see that OR search spaces can be thought of as and AND/OR search space guided by a pseudo tree with no branching variables (also known as a chain pseudo tree) with the search space omitting explicit OR nodes (which are instead implicitly captured by the various levels in the search space). For example, the search space from Figure 4 can be explicitly represented as the AND/OR search space shown in Figure 8.

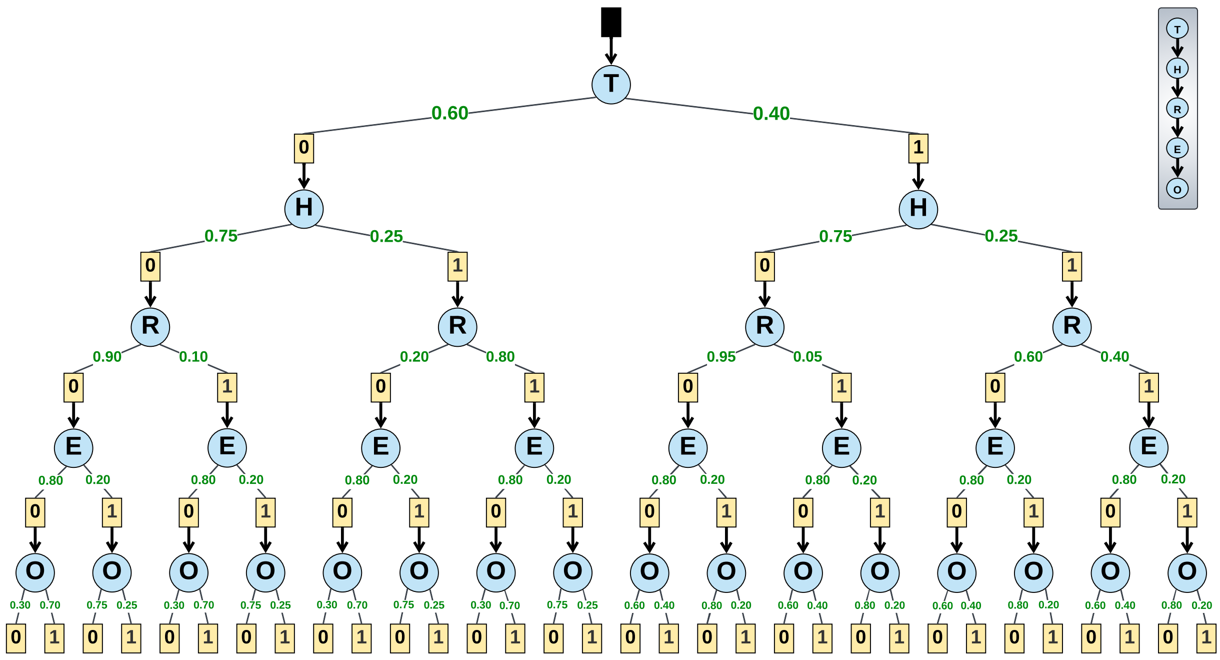


Figure 8: The OR search space from Figure 4 expressed explicitly as an AND/OR search space.

2.1.3 Search Space Notation

The symbol n is used to generally represent search nodes. (Depending on context, n may represent either AND or OR nodes). n_X specifically refers to an AND node in and AND/OR search tree T associated with variable X. Y_{n_X} represents the specific OR node associated with variable Y that is the child of n_X. The notation of OR nodes may seem counter intuitive at first as they resemble the notation for variables of a graphical model. However, this is because OR nodes of AND/OR trees representing graphical models in fact do represent the variables of the model which explains the choice for its notation. $path(n)$ is the configuration of the variables along the path from the root of T to n according to assignments corresponding to that path. For example, in Figure 7a, if we let n be the leaf node of the highlighted path, $path(n) = \{T = 0, H = 1, R = 0\}$. $varpath(n)$ is the set of variables that $path(n)$ provides a configuration for. With this notation, we can express the cost of the arc to an AND node n_X as:

$$c(n_X) = \prod_{f_\alpha | varpath(n_X) \subseteq \alpha \text{ and } \{X\} \subseteq \alpha} f_\alpha(path(n_X)) \tag{10}$$

$g(n)$ is the cost of $path(n)$ according to the combination operation defined for the model. For example, $g(n)$ for the same leaf node n in Figure 7a assuming a product combination operation would be $g(n) = (0.6) \cdot (0.25) \cdot (0.20)$. $ch(n)$ denotes the children of node n. Note that for AND/OR nodes, children of AND nodes are OR nodes, and vise versa. $anc(n)$ are all the ancestors of n. In AND/OR trees, $br(n_X)$ is the set of ancestor AND nodes n_Y on the path to n_X such that Y is a branching variable in $\mathcal{T}$.

2.1.4 Important Quantities in AND/OR Search Spaces

To use AND/OR search spaces effectively for solving tasks, there are several quantities that become important to compute and understand. We will describe these next.

***Z*(*n*).**
As we saw in Section 1.6.1: Task Formalizations, we use Z to denote the partition function of a model - namely the sum of costs across all configurations of the model. Each configuration's cost is calculated as the product of the model's functions based on the assignments corresponding to that particular configuration. In the context of AND/OR search, given infinite resources Z could be computed by systematic search enumerating all full configurations and summing their costs (see Figure 7b for an example full configuration).

Applying a similar concept, we introduce the quantity $Z(n)$. Semantically, $Z(n)$ represents the partition function of an imaginary model that could be represented by the subtree rooted at n. Thus $Z(n)$ is equal to the sum of the costs of all partial configurations rooted at n due to only functions that contribute to the arc values under n. For an AND node n_X with children OR nodes $Y_{n_X} \in ch(n_X)$, $Z(n_X)$ can be computed by

$$Z(n_X) = \prod_{Y_{n_X} \in ch(n_X)} Z(Y_{n_X}) \tag{11}$$

such that for OR nodes Y_{n_X}, $Z(Y_{n_X})$ is computed by

$$Z(Y_{n_X}) = \sum_{n_Y \in ch(Y_{n_X})} c(n_Y) \cdot Z(n_Y) \tag{12}$$

with $Z(n_X) = 1$ vacuously, in the case it has no children.

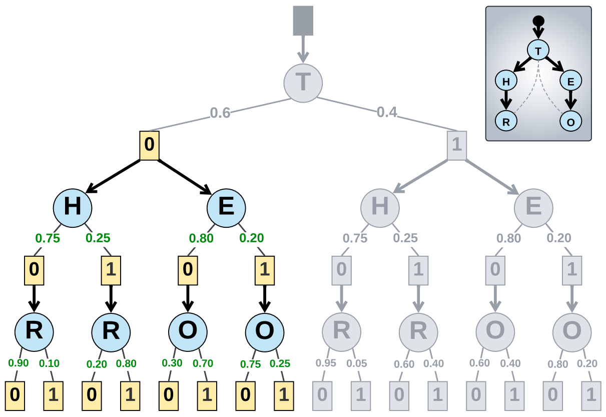


Figure 9: The subtree contributing to $Z(n_{T=0})$ is highlighted above. Using Equation 11 and Equation 12, $Z(n_{T=0}) = 1.0$.

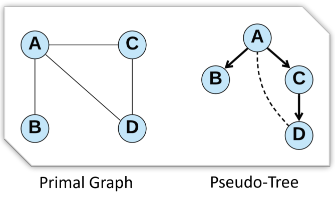
Note that given $n_\varnothing$ as the dummy root node of AND/OR tree T, $Z(n_\varnothing) = Z$ of the underlying model $\mathcal{M}$. We denote estimation of $Z(n)$ as $\hat{Z}(n)$. Heuristic estimates of $Z(n)$ are denoted as $h(n)$.

***R*(*n*).** On the path from the root of an AND/OR tree T to some node n_X, there may be an intermediate node n_Y associated with branching variable Y in the guiding pseudo tree $\mathcal{T}$. (For example in Figure 10b, on the path to the highlighted node $n_{A=0,C-1}$, node $n_{A=0}$ is traversed where A is a branching variable in Figure 10). When this happens, the remaining variables of the model are split between different branches. (For example in the same Figure 10b, notice the left branch under the node $n_{A=0}$ contains variable B but not C or D and that the right branch contains C and D but not B). Thus, the $Z(n)$ of any node down one of the branches will necessarily miss the cost from the configurations of the variables included in the other branch(es). $R(n_X)$, or the *ancestor branching mass*, captures the $Z(n)$ for all variables that had branched off of the path to n_X. (For example in the same Figure 10b, the green box shows the portion corresponding to the $R(n_{A=0,C=1}) = Z(B_{n_{A=0}})$).

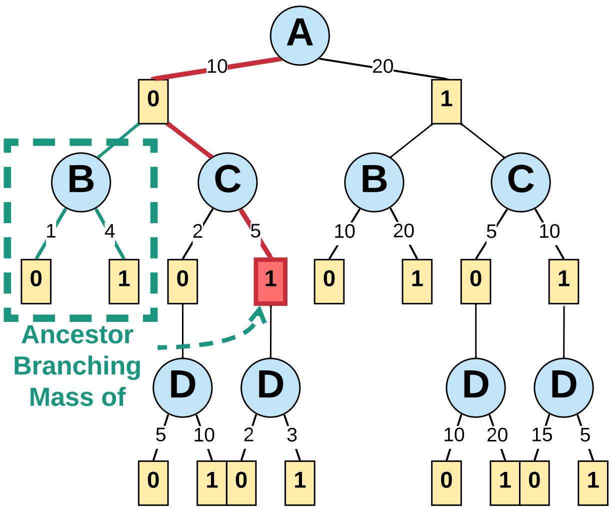
More formally, let $br(n_X)$ be the set of ancestor nodes n_Y on the path to n_X such that Y is a branching variable in $\mathcal{T}$. Let W_{n_Y} be the child OR node of n_Y that that is also on the path to X. We define $R(n_X)$ as:

$$R(n_X) = \prod_{n_Y \in br(n_X)} \frac{Z(n_Y)}{Z(W_{n_Y})} \tag{13}$$

We denote approximations to $R(n)$ as $r(n)$.



(a)



(b)

Figure 10: A full AND/OR tree representing 16 possible solutions guided by the pseudo tree shown above. Boxed in green is the ancestor branching subtree for the path $\rightarrow(A{=}0)\rightarrow(C{=}1)$.

***$Q(n)$*.** We can now concisely define a quantity $Q(n)$ as the contribution to Z from all full configurations consistent with $path(n)$. In other words, $Q(n)$ is the likelihood of the configuration $path(n)$ based on the distribution defined by $\mathcal{M}$, with $P(path(n)) = \frac{Q(n)}{Z}$. $Q(n)$ can be computed simply as:

$$Q(n) = g(n)\cdot R(n)\cdot Z(n) \tag{14}$$

Example. In Figure 10b, consider the path from the root to the red node $n_{A=0,C=1}$. Following $n_{A=0}$ to our node, we see OR node $B_{n_{A=0}}$ that branches off of the path. So,

$$\begin{aligned} Q(n_{A=0,C=1}) &= g(n_{A=0,C=1})\cdot R(n_{A=0,C=1})\cdot Z(n_{A=0,C=1}) \\ &= g(n_{A=0,C=1})\cdot Z(B_{n_{A=0}}) \quad \cdot Z(n_{A=0,C=1}) \\ &= (10\cdot 5) \qquad\quad \cdot(1\cdot 1 + 4\cdot 1) \quad \cdot(2\cdot 1 + 3\cdot 1) \end{aligned}$$

2.2 INFERENCE

Next we will outline key inference schemes used with probabilistic graphical models relevant to this work.

2.2.1 Variable Elimination

Many probabilistic graphical model queries can be solved by an inference framework known as **variable elimination** (**VE**). Variable elimination involves an ordered computational processing of the variables of a model, at each step removing a processed variable from subsequent computations (thus called an *elimination* step). Each elimination step corresponds to a step of inference, transferring the effects of the eliminated variables over to the remaining variables (in practice, done by creating a newly inferred function over the remaining variables).

As an example, consider the query to find the mode of the distribution defined by our simple example above (Section 1.4:

Simple Example). Formalizing this query, we want to solve the task:

$$\max_{T,H,E,R,O} p(t,h,e,r,o) \tag{15}$$

which, based on Equation 5, in terms of our model is equivalent to

$$\max_{T,H,E,R,O} p(t,h,e,r,o) = \max_{T,H,E,R,O} f_T(t) \cdot f_H(h) \cdot f_E(e) \cdot f_{T,H,R}(t,h,r) \cdot f_{T,E,O}(t,e,o) \tag{16}$$

(The blue coloring is simply to help keep note of where the functions are in the expression).

Using variable elimination to solve this query, we would first need an **elimination order** - order in which to process and eliminate variables while performing inference. Suppose an elimination order $o_{elim} = [R, O, E, H, T]$. Then, given this ordering, we express our query as

$$\max_T(\ \max_H(\ \max_E(\ \max_O(\ \max_R(\ f_T(t) \cdot f_H(h) \cdot f_E(e) \cdot f_{T,H,R}(t,h,r) \cdot f_{T,E,O}(t,e,o)\ )\ )\ )\ )\ ) \tag{17}$$

where the query can then be solved inside-to-out, variable-by-variable, via computations indicated by the parenthesis. The result from each step can be interpreted as the inference performed over its corresponding variable.

One power of variable elimination is its ability to simplify computation leveraging mathematical properties of the query. Note that in our example some of the model's functions are not dependent on the variable being immediately maximized over and so can be factored out of the respective maximization. Doing so recursively, we can rewrite our query with the same ordering instead as

$$\max_T(\ f_T(t) \cdot \max_E(\ f_E(e) \cdot \max_O(\ f_{T,E,O}(t,e,o)\ )\ ) \cdot \max_H(\ f_H(h) \cdot \max_R(\ f_{T,H,R}(t,h,r)\ )\ )\ ) \tag{18}$$

This decomposition reduces the size of the terms being maximized over, thus reducing complexity of the computations.

2.2.2 Bucket Elimination

Bucket elimination Dechter [1999], or **BE**, is a variable elimination scheme that can be adapted for a myriad of graphical model tasks including those described in Section 1.6: Some Well-Known and Important Graphical Model Tasks.

Bucket elimination is a message passing scheme that performs variable elimination according to a given elimination order by processing a data structure called **buckets** one-by-one, each bucket corresponding to a variable in the ordering. When reaching a variable X_i in the ordering, all unprocessed functions that contain X_i in their scope are placed in bucket B_i (this includes the model's original functions as well as messages generated during the bucket elimination process). As shown in Equation 19, the bucket is then processed by applying an elimination operation (generalized as $\bigoplus$) over X_i to the combination of the bucket functions (generalized as $\bigotimes$) resulting in a **bucket message** - a new function over the remaining variables present in the scope of the processed functions - denoted $\lambda_{i\to j}$, or λ_i for short.

$$\lambda_{i\to j} = \bigoplus_{X_i} \bigotimes_{f_\alpha \in B_i} f_\alpha(\alpha) \tag{19}$$

In the context of computing the **partition function**, this corresponds to marginalizing X_i from the product of the functions

$$\lambda_{i\to j} = \sum_{X_i} \prod_{f_\alpha \in B_i} f_\alpha(\alpha) \tag{20}$$

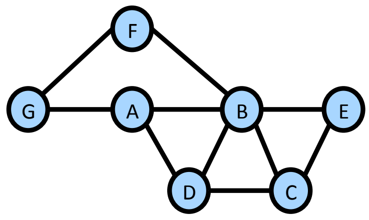
or in the context of computing the **MAP**, maximizing the product of the functions over X_i

$$\lambda_{i\to j} = \max_{X_i} \prod_{f_\alpha \in B_i} f_\alpha(\alpha) \tag{21}$$

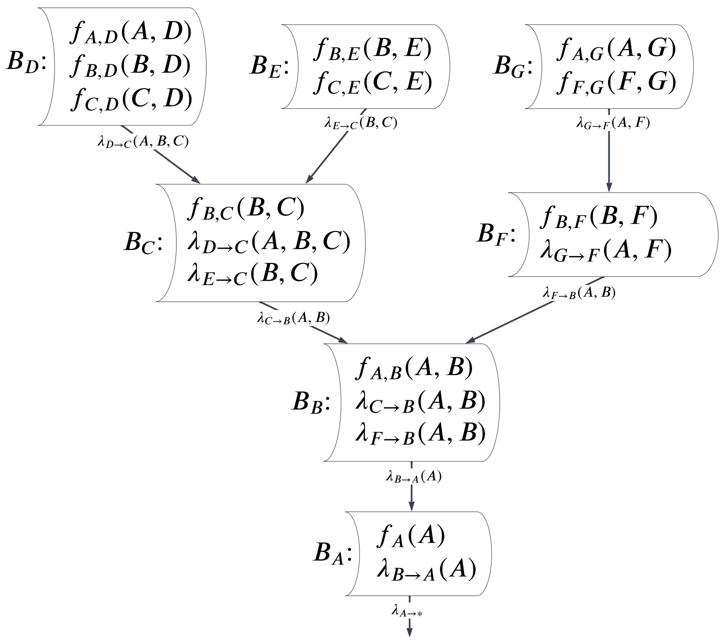
The i in $\lambda_{i\to j}$ refers to the bucket that generated the message. j indicates the bucket this message will be sent to; namely the next variable in the elimination ordering that is also found in the scope of the message.

The processed buckets and messages can then be used to compute result of the underlying query (ex. in the case of computing the partition function or MAP, the result is simply the combination of the finally remaining messages after processing of the last bucket). Figure 11 shows a schematic of bucket elimination on a graphical model with variables indexed from A to G and with a unary function with respect to variable A and pair-wise functions over the pairs of variables connected by an edge in the underlying primal graph (Figure 11a), namely: $F = \{f_A(A), f_{A,B}(A,B), f_{A,D}(A,D), f_{A,G}(A,G), f_{B,C}(B,C), f_{B,D}(B,D), f_{B,E}(B,E), f_{B,F}(B,F), f_{C,D}(C,D), f_{C,E}(C,E), f_{F,G}(F,G)\}$.

Bucket elimination can be viewed as a 1-iteration message-passing algorithm along its **bucket tree** (Figure 11b). The nodes of the tree are the different buckets. Each bucket of a variable contains a set of the model's functions depending on the given



(a) Example primal graph of a graphical model with 7 variables and model functions $F = \{f_A(A),\ f_{A,B}(A,B),\ f_{A,D}(A,D),\ f_{A,G}(A,G),\ f_{B,C}(B,C),\ f_{B,D}(B,D),\ f_{B,E}(B,E),\ f_{B,F}(B,F),\ f_{C,D}(C,D),\ f_{C,E}(C,E),\ f_{F,G}(F,G)\}$.



B_D: $f_{A,D}(A,D)$ $f_{B,D}(B,D)$ $f_{C,D}(C,D)$
B_E: $f_{B,E}(B,E)$ $f_{C,E}(C,E)$
B_G: $f_{A,G}(A,G)$ $f_{F,G}(F,G)$
$\lambda_{D\to C}(A,B,C)$
$\lambda_{E\to C}(B,C)$
$\lambda_{G\to F}(A,F)$
B_C: $f_{B,C}(B,C)$ $\lambda_{D\to C}(A,B,C)$ $\lambda_{E\to C}(B,C)$
B_F: $f_{B,F}(B,F)$ $\lambda_{G\to F}(A,F)$
$\lambda_{C\to B}(A,B)$
$\lambda_{F\to B}(A,B)$
B_B: $f_{A,B}(A,B)$ $\lambda_{C\to B}(A,B)$ $\lambda_{F\to B}(A,B)$
$\lambda_{B\to A}(A)$
B_A: $f_A(A)$ $\lambda_{B\to A}(A)$
$\lambda_{A\to *}$

(b) Bucket elimination schematic following an elimination order $o_{elim} = [D,E,G,C,F,B,A]$.

Figure 11: (a) A primal graph of a graphical model with 7 variables. (b) Illustration of *BE* with an ordering A B C E D F G.

order of processing. There is an arc from bucket B_X to bucket B_Y, if the function created at bucket B_X is placed in bucket B_Y.

In summary, bucket elimination uses the variable elimantion paradigm and dynamic programming to break a computational task into smaller subproblems, computing the result by processing buckets and sending resulting messages according to a provided elimination order.

Complexity. Both the time and space complexity of bucket elimination are exponential in the **induced width** of the model, which can be computed as a graph parameter based on the provided ordering and the underlying primal graph Dechter [2019]. (More on the induced width in Section 2.2.3: Induced Width (w*)). In the context of bucket messages, the induced width is equal to the cardinality of the scope of the bucket message with the largest scope. Given that its complexity is exponential in the induced width, bucket elimination becomes impractical if the induced width is large, and thus approximation schemes have been developed to address this Dechter and Rish [2002], Liu and Ihler [2011].

2.2.3 Induced Width (w*)

The difficulty of answering a query using variable elimination can depend heavily on the elimination order being used, with some elimination orderings leading to efficient factorization, whereas others may not and instead result large computations. We can capture this complexity graphically.

As elimination computations are performed variable-by-variable pursuant to the ordering provided, the corresponding solutions (which themselves may be a function over some remaining variables) can be viewed as inducing new edges onto the underlying primal graph in the same way the model's native functions did originally (see Section 1.3: Primal Graph for details). These new edges correspond to newly inferred relationships between variables not directly connected in the original graph. When adding all of the newly induced edges to the primal graph (namely the new edges resulting from generated messages from the processing of all the variables), we end up with a new graph called the ***induced primal graph***, or ***induced graph*** for short. The complexity of exact variable elimination algorithms are with respect to the tree-width of the resulting induced graph - namely with respect to a quantity known as the ***induced width*** (or ***w****), which is one less than the largest clique-size of the induced graph.

Definition 2.2.3.1 (Induced Width (w*)). *The induced width of a graphical model $\mathcal{M}$ with respect to elimination order o_{elim} with induced primal graph $\mathcal{G}'$ is*

$$w^* = \max_{c \in clq(\mathcal{G}')} |c| - 1, \tag{22}$$

where $clq(\mathcal{G}')$ is the set of all cliques in $\mathcal{G}'$ and $|c|$ denotes the clique-size of clique c.

References

A. Darwiche. *Modeling and Reasoning with Bayesian Networks*. Cambridge University Press, 2009.

Rina Dechter. Bucket elimination: A unifying framework for reasoning. *Artificial Intelligence*, 113:41–85, 1999.

Rina Dechter. *Reasoning with Probabilistic and Deterministic Graphical Models: Exact Algorithms*. Synthesis Lectures on Artificial Intelligence and Machine Learning. Morgan & Claypool Publishers, 2013. doi: 10.2200/S00529ED1V01Y201308AIM023. URL http://dx.doi.org/10.2200/S00529ED1V01Y201308AIM023.

Rina Dechter. Reasoning with probabilistic and deterministic graphical models: Exact algorithms, second edition. *Synthesis Lectures on Artificial Intelligence and Machine Learning*, 13:1–199, 02 2019.

Rina Dechter and Robert Mateescu. AND/OR search spaces for graphical models. *Artificial Intelligence*, 171(2-3):73–106, 2007.

Rina Dechter and I Rish. Mini-buckets: A general scheme for approximating inference. *Journal of the ACM*, pages 107–153, 2002.

Qiang Liu and Alexander Ihler. Bounding the partition function using Hölder's inequality. In *International Conference on Machine Learning (ICML)*, pages 849–856. ACM, June 2011.

J. Pearl. *Probabilistic Reasoning in Intelligent Systems*. Morgan Kaufmann, 1988.