Even more generic solution construction in Valuation-Based Systems

Jordi Roca-Lacostena   Jesus Cerquides [*]

February 27, 2014

Abstract

Valuation algebras abstract a large number of formalisms for automated reasoning and enable the definition of generic inference procedures. Many of these formalisms provide some notions of solutions. Typical examples are satisfying assignments in constraint systems, models in logics or solutions to linear equation systems.

Recently, formal requirements for the presence of solutions and a generic algorithm for solution construction based on the results of a previously executed inference scheme have been proposed [9, 8]. Unfortunately, the formalization of Pouly and Kohlas relies on a theorem for which we provide a counter example. In spite of that, the mainline of the theory described is correct, although some of the necessary conditions to apply some of the algorithms have to be revised. To fix the theory, we generalize some of their definitions and provide correct sufficient conditions for the algorithms. As a result, we get a more general and corrected version of the theory presented at [9, 8].

1   Introduction

Solving discrete optimization problems is an important and well-studied task in computer science. One particular approach to tackle them is known as dynamic programming [2] and can be found in almost every handbook about algorithms and programming techniques. The works of Bellman [1], Nemhauser [7] and Bertelè and Brioschi [2] present non-serial dynamic programming as an algorithm for optimization problems for functions taking values in the real numbers. A more general approach was taken by Mitten [6] and further generalized by Shenoy in 1996 [10], for functions taking values in any ordered set Δ. Shenoy introduces a set of axioms that later on will be known as valuation algebras. In those terms, Shenoy is the first one to connect the concept of solution with the marginalization operation of the valuation algebra.

In 2011, Pouly and Kohlas [9, 8] drop the assumption that valuations are functions that map tuples into a value set Δ. They present several algorithms,

---

[*]IIIA - CSIC, Campus UAB, Spain, email: {jroca, cerquide}@iiia.csic.es

and characterize the sufficient conditions for its correctness. Pouly and Kohlas' algorithms are more general than their predecessors in the literature. This increased generality comes at no computational cost, since when applied in the previously covered scenarios, their particularization coincides exactly with the previously proposed algorithm. Furthermore, by dropping the assumption that valuations are functions, their algorithms can be applied to previously uncovered cases such as the solution of linear equation systems or the algebraic path problem. Unfortunately, one of the fundamental results in Pouly and Kohlas' theory is incorrect.

The contributions of this work are:

1. We provide a counterexample that invalidates Pouly and Kohlas' results.

2. We generalize the problem solved by Pouly and Kohlas, and provide and algorithm to solve it.

3. We provide a new sufficient condition for the correctness of the algorithm.

These results provide the most general theory for dynamic programming up-to-date.

2 Background

In this section we start by defining valuation algebras. Later on, we introduce the marginalization problem and finally we review the COLLECT algorithm to solve that problem.

The basic elements of a valuation algebra are so-called *valuations*, that we subsequently denote by lower-case Greek letters such as ϕ or ψ. Let D be a lattice[4] with a partial order $\leq$, two operations meet $\wedge$ and join $\vee$, a top element $\top$, and a bottom element $\bot$. Given a set of valuations Φ, and a lattice of domains D, a valuation algebras has three operations:

1. *Labeling:* $\Phi \to D; \phi \mapsto d(\phi)$,

2. *Combination:* $\Phi \times \Phi \to \Phi; (\phi, \psi) \mapsto \phi \otimes \psi$,

3. *Projection:* $\Phi \times D \to \Phi; (\phi, x) \mapsto \phi^{\downarrow x}$ for $x \leq d(\phi)$.

satisfying the following axioms:

A1 *Commutative semigroup*: Φ is associative and commutative under $\otimes$.

A2 *Labeling:* For $\psi, \phi \in \Phi$, $d(\phi \otimes \psi) = d(\phi) \vee d(\psi)$.

A3 *Projection:* For $\phi \in \Phi$, $x \in D$, and $x \leq d(\phi)$, $d(\phi^{\downarrow x}) = x$.

A4 *Transitivity:* For $\phi \in \Phi$ and $x \leq y \leq d(\phi)$, $(\phi^{\downarrow y})^{\downarrow x} = \phi^{\downarrow x}$.

A5 *Combination:* For $\phi, \psi \in \Phi$ with $d(\phi) = x$, $d(\psi) = y$, and $z \in D$ such that $x \leq z \leq x \vee y$, $(\psi \otimes \phi)^{\downarrow z} = \phi \otimes \psi^{\downarrow z \wedge y}$.

A6 *Domain:* For $\phi \in \Phi$ with $d(\phi) = x$, $\phi^{\downarrow x} = \phi$.

We say that valuation $e \in \Phi$ is an identity valuation provided that $d(e) = \bot$ and $\phi \otimes e = \phi$ for each $\phi \in \Phi$. As proven in [9], any valuation algebra that does not have and identity valuation can easily be extended to have one. In the following and without loss of generality we assume that our valuation algebra has an identity valuation e.

Variable systems, frames and tuples. In most practical applications of valuation algebras, the domains of the valuations are subsets of a given set of variables V. It is well known (e.g. [3] page 36) that for any set V, the ordered set $\langle \mathcal{P}(V), \subseteq \rangle$ is a complete lattice, referred to as the *power set lattice.* Thus, most of the work on valuation algebras assumes that the lattice D is the power set lattice of a set V of variables.

Let $V = \{x_1, \ldots, x_n\}$ be a finite set of variables[1]. We assume that for each variable $x \in V$ we can assign a set Ω_x of possible values, called its *frame.* Similarly, the frame of $X \subseteq V$ is $\Omega_X = \prod_{x \in X} \Omega_x$. It is mathematically convenient to include a singleton element (noted as $\diamond$) in $\Omega_\emptyset$. Thus, $\Omega_\emptyset = \{\diamond\}$. A pair $\langle V, \Omega \rangle$ is known as a *variable system.* In many cases a variable system is naturally linked to a valuation algebra. Then we say that the valuation algebra is equipped with a variable system. A typical example is when valuations are discrete real functions and the domain of a valuation is the set of discrete variables over which the function is defined.

A *tuple* $\mathbf{x}$ with finite domain $X \subseteq V$ is an element of Ω_X. A projection operation can be defined on tuples, unrelated to the projection operation of the valuation algebra. Given a tuple $\mathbf{x}$ with domain X and $Y \subseteq X$ we define the projection of $\mathbf{x}$ to Y as the tuple $\mathbf{y}$ that results from $\mathbf{x}$ by discarding the values of the variables in $X - Y$. We note the projection of $\mathbf{x}$ to Y as $\mathbf{x}^{\downarrow Y}$. We can write $\mathbf{x} = (\mathbf{x}^{\downarrow Y}, \mathbf{x}^{\downarrow X - Y})$. Furthermore, $\mathbf{x} = (\mathbf{x}, \diamond) = (\diamond, \mathbf{x})$.

Example 1. Given a set of binary variables V, we consider its power set lattice as the domain lattice of the valuation algebra. The set of valuations is composed by all the functions $\phi : \Omega_X \to \{0, 1\}$, where $X \subseteq V$. The labeling operation is defined by $d(\phi) = X$. The combination of two valuations ϕ, ψ, is the valuation $(\phi \otimes \psi)(\mathbf{x}) = \phi(\mathbf{x}^{\downarrow d(\phi)}) + \psi(\mathbf{x}^{\downarrow d(\psi)})$, whereas the projection of a valuation ϕ with $d(\phi) = X$ to a domain $Y \subseteq X$ is the valuation $\phi^{\downarrow Y}(\mathbf{y}) = \max_{\mathbf{z} \in \Omega_{X-Y}} \phi(\mathbf{y}, \mathbf{z})$. As proven in [9] the valuation algebra of Boolean functions satisfies axioms A1-A6.

Some other relevant examples of valuation algebras are relational algebra, which is fundamental to databases, or the algebra of probability potentials, which underlies many results in probabilistic graphical models and the more abstract class of semiring induced valuation algebras [5].

---

[1]All the definitions are correct not only for finite but also for countable V

2.1 Finding the marginal of a factorized valuation

A relevant problem in many valuation algebras in the problem of finding the marginal of a factorized valuation.

Problem 1. Let (Φ, D) be a valuation algebra, and $\phi_1, \ldots, \phi_n$ be valuations in Φ. Find $(\phi_1 \times \cdots \times \phi_n)^{\downarrow X}$.

Note that when our valuations are probability potentials, this is the well studied problem of finding the marginal of a factorized distribution, also known as Markov Random Field.

The FUSION algorithm [10] (a.k.a. variable elimination) or the COLLECT algorithm (a.k.a. junction tree or cluster tree algorithm)[9, 8] can be used to find marginals. Since our results build on top of the COLLECT algorithm, we provide a more accurate description below.

A necessary condition to apply the COLLECT algorithm is that we can organize the valuations $\phi_1, \ldots, \phi_n$ into a covering join tree, which we introduce next.

Definition 1. A labeled tree is any tree (V, E) together with a function $\lambda : V \to D$ that links each node with a single domain in D.

A join tree is a labeled tree $\mathcal{T} = (V, E, \lambda, D)$ such that for any $i, j \in V$ it holds that $\lambda(i) \wedge \lambda(j) \leq \lambda(k)$ for all nodes k on the path between i and j. In that case, we say that $\mathcal{T}$ satisfies the running intersection property.

Definition 2. Given a valuation $\phi = \phi_1 \times \cdots \times \phi_n$ we say that a join tree $\mathcal{T} = (V, E, \lambda, D)$ is a covering join tree for this factorization if $|V| = n$ and for all ϕ_i there is a node $j \in V$ such that $d(\phi_i) \leq \lambda(j)$.

Definition 3. Let i be a node in a rooted junction tree whose root is r. We use $p(i)$ to denote the parent of i in the tree. The separator of i is $s_i = \begin{cases} \perp & \text{, if } i = r \\ \lambda(i) \wedge \lambda(p(i)) & \text{, otherwise} \end{cases}$

Algorithm 1 provides a description of the COLLECT algorithm. It is based on sending messages up the tree, through the edges of the covering join tree, until the root node is reached. The message sent from each node summarizes the information in the corresponding subtree which is relevant to its parent. The running intersection property guarantees that no information is lost.

Theorem 1. *After running the COLLECT algorithm (Algorithm 1) over the nodes of a covering join tree for $\phi = \prod_k \phi_k$, we have that $\psi_i' = \left(\prod_{j \in \mathcal{T}_i} \psi_j \right)^{\downarrow \lambda(i)}$. In particular, if r is the root $\psi_r' = \phi^{\downarrow \lambda(r)}$.*

The theorem is an adaptation of Theorem 3.6 in [9]. As a consequence of this theorem, we can use the COLLECT algorithm to solve problem 1 provided that we are given a covering join tree for the factorization we would like to marginalize.

Algorithm 1 COLLECT algorithm

Each node i of the junction tree executes

1: ▷ Assess the product of valuations assigned to i
2: $\psi_i \leftarrow e \times \prod_{j \in a^{-1}(i)} \phi_j$
3: From each child j of i, receive a message $\mu_{j \rightarrow i}$.
4: ▷ Incorporate messages from children
5: $\psi_i' \leftarrow \psi_i \times \prod_j \mu_{j \rightarrow i}$.
6: **if** i is not the root **then**
7:     Send message $\psi_i^{\downarrow s_i}$ to its parent $p(i)$
8: **end if**

---

3 Generic solutions in valuation algebras with variable system

In the previous section we have shown that the COLLECT algorithm can be used to find marginals. In this section we focus on the *solution finding problem* (SFP).

The problem is of foremost importance, since it lies at the foundation of dynamic programming [10, 2]. Furthermore, problems such as satisfiability, solving Maximum a Posteriori queries in a probabilistic graphical models, or maximum likelihood decoding are particular instances of the SFP.

We start by formally defining the problem and then we review the results of Pouly [8] and Pouly and Kohlas [9], who give algorithms for solving those problems and establish the conditions under which those algorithms are guaranteed to work. Unfortunately, although the inspirational ideas underlying Pouly and Kohlas' work are correct, their formal development is not. Thus, we end up the section providing a counter example to one of their main theorems.

3.1 The SFP for valuation algebras with variable system

Up to know, the most general formalization of the SFP is the one provided by [10] and adapted by Pouly and Kohlas to the formal framework of valuation algebras in Chapter 8 of [9]. They assume a valuation algebra $\langle \Phi, D \rangle$ equipped with a variable system $\langle V, \Omega \rangle$. As in the marginal assessment problem, in the SFP, we are given a set of valuations $\phi_1, \ldots, \phi_m \in \Phi$ as input. However, instead of a marginal of its combination $\phi = \phi_1 \times \ldots \times \phi_m$, we are required to provide a tuple $\mathbf{x}$ with domain $d(\phi)$, such that $\mathbf{x}$ is a solution for ϕ. In order for the previous sentence to make any sense we need to properly define our concept of *solution*. The most general way in which we can do this is by defining a family $c = \{c_\phi | \phi \in \Phi\}$ of solution sets, such that for each valuation $\phi \in \Phi$, the solution set c_ϕ contains the subset of $\Omega_{d(\phi)}$ such that $\mathbf{x} \in \Omega_{d(\phi)}$ is considered a solution for ϕ if and only if $\mathbf{x} \in c_\phi$. We say that the family of sets c is a *solution concept*. Now we can formally define the SFP as follows

Problem 2 (SFP with variable system)**.** Given a valuation algebra $\langle \Phi, D \rangle$ equipped with a variable system $\langle V, \Omega \rangle$ and a solution concept c, and a set

of valuations $\phi_1, \ldots, \phi_m \in \Phi$, find $\mathbf{x} \in \Omega_{d(\phi)}$ such that $\mathbf{x}$ is a solution for $\phi = \phi_1 \times \ldots \times \phi_m$.

3.2 Solving the solution finding problem by composing partial solutions

Several authors have provided algorithms that solve the SFP and characterized under which conditions they can be successfully applied.

As described in the introduction, several works have sought to provide a formal foundation to dynamic programming, which we can now identify as a particular case of the SFP. In their works in 2011, Pouly and Kohlas [9, 8] drop the assumption that valuations are functions that map tuples into a value set Δ. They present several algorithms, and characterize the sufficient conditions for its correctness. By dropping the assumption that valuations are functions, their algorithms can be applied to previously uncovered cases .

Essentially, the sufficient conditions for the correctness of Pouly and Kohlas' algorithms connect the operations in the valuation algebra with the solution concept by means of a *family of configuration extension set*s.[2] A family of configuration extension sets $\mathcal{W}$ assigns a configuration extension set to each pair $\langle \phi, \mathbf{x} \rangle$ such that ϕ is a valuation and $\mathbf{x}$ is a tuple whose domain X is a subset of $d(\phi)$. That is $\mathcal{W} = \{W_\phi(\mathbf{x}) \subseteq \Omega_{d(\phi)-X} | \phi \in \Phi, X \subseteq d(\phi), \mathbf{x} \in \Omega_X\}$. Furthermore, a family of configuration extension sets has to satisfy two conditions. The first one connects the projection operation of the valuation algebra with the extension sets by imposing that every extension set can be calculated in two steps. The second one connects the set of solutions of a valuation with the set of extensions of the empty tuple $\diamond$. These conditions can be stated formally as follows

1. For each $\phi \in \Phi$, for each $X \subseteq Y \subseteq d(\phi)$ and for each $\mathbf{x} \in \Omega_X$ we have that $W_\phi(\mathbf{x}) = \{\mathbf{z} \in \Omega_{d(\phi)-X} \mid \mathbf{z}^{\downarrow Y-X} \in W_{\phi^{\downarrow Y}}(\mathbf{x})$ and $\mathbf{z}^{\downarrow d(\phi)-Y} \in W_\phi(\mathbf{x}, \mathbf{z}^{\downarrow Y-X})\}$.

2. For each $\phi \in \Phi$, $c_\phi = W_\phi(\diamond)$.

Based on this definition, Pouly and Kohlas state the following theorem

Theorem 2 (Theorem 8.1 in [9]). *For any valuation $\phi \in \Phi$ and any $X, Y \subseteq d(\phi)$, we have*

$$c_\phi^{\downarrow X \cup Y} = \{\mathbf{z} \in \Omega_{X \cup Y} \mid \mathbf{z}^{\downarrow Y} \in c_\phi^{\downarrow Y} \ and \ \mathbf{z}^{\downarrow X-Y} \in W_{\phi^{\downarrow X}}(\mathbf{z}^{\downarrow X \cap Y})\}. \tag{1}$$

Unfortunately, the theorem is not correct. We will use the valuation algebra on Boolean functions from example 1to build a counterexample for it. Pouly and Kohlas (equation 8.19) defined the extension sets for the algebra of Boolean lattices as $W_\phi(\mathbf{x}) = \{\mathbf{y} \in \Omega_{d(\phi)-X} \mid \phi(\mathbf{x}, \mathbf{y}) = \phi^{\downarrow X}(\mathbf{x})\}$, where $X \subseteq d(\phi)$, $\mathbf{x} \in \Omega_X$.

---

[2] Although they do never formally introduce families of configuration extension sets, we have introduced the concept here for mathematical correctness and so that the reader can easily follow the generalization that will come later on.

Counterexample 1. *Taking ϕ as the Boolean function* $\phi(\mathbf{x},\mathbf{y}) = \begin{cases} 1 & \text{if } \mathbf{x} = \mathbf{y}, \\ 0 & \text{otherwise} \end{cases}$*,*
$X = \{x\}$*, and* $Y = \{y\}$ *the result in theorem 2 does not hold.*

To see why, we can assess both sides of equation 1 and see that they are not the same. For the left hand side, note that $c_\phi^{\downarrow X \cup Y} = c_\phi = W_\phi(\diamond)$. Thus, $c_\phi = \{(\mathbf{x},\mathbf{y}) \in \Omega_{X \cup Y} \mid \phi(\mathbf{x},\mathbf{y}) = \phi^{\downarrow\emptyset}(\diamond)\}$. We can assess $\phi^{\downarrow\emptyset}(\diamond) = \max_{x.y} \phi(\diamond,(\mathbf{x},\mathbf{y})) = \max_{x,y} \phi(\mathbf{x},\mathbf{y}) = 1$. Hence, $c_\phi^{\downarrow X \cup Y} = c_\phi = \{(0,0),(1,1)\}$.

Let A denote the set at the r.h.s. of equation 1. Since $X \cap Y = \emptyset$ and $X - Y = X$, we have that $A = \{\mathbf{z} \in \Omega_{X \cup Y} \mid \mathbf{z}^{\downarrow Y} \in c_\phi^{\downarrow Y} \text{ and } \mathbf{z}^{\downarrow X-Y} \in W_{\phi^{\downarrow X}}(\mathbf{z}^{\downarrow X \cap Y})\} = \{\mathbf{z} \in \Omega_{X \cup Y} \mid \mathbf{z}^{\downarrow Y} \in c_\phi^{\downarrow Y} \text{ and } \mathbf{z}^{\downarrow X} \in W_{\phi^{\downarrow X}}(\diamond)\}$. We can assess $c_\phi^{\downarrow Y} = \{\mathbf{z}^{\downarrow Y} \mid \mathbf{z} \in c_\phi\} = \{(0),(1)\}$. Furthermore, we have that $W_{\phi^{\downarrow X}}(\diamond) = \{\mathbf{x} \in \Omega_X \mid \phi^{\downarrow X}(\mathbf{x}) = (\phi^{\downarrow X})^{\downarrow\emptyset}(\diamond)\} = \{\mathbf{x} \in \Omega_X \mid \phi^{\downarrow X}(\mathbf{x}) = \phi^{\downarrow\emptyset}(\diamond)\} = \{\mathbf{x} \in \Omega_X \mid \phi^{\downarrow X}(\mathbf{x}) = 1\}$. We have that for all $\mathbf{x} \in X$, $\phi^{\downarrow X}(\mathbf{x}) = \max_{\mathbf{y}} \phi(\mathbf{x},\mathbf{y}) = 1$, and thus, $W_{\phi^{\downarrow X}}(\diamond) = \Omega_X = \{0,1\}$. Hence, $A = \{\mathbf{z} \in \Omega_{X \cup Y} \mid \mathbf{z}^{\downarrow Y} \in \Omega_Y \text{ and } \mathbf{z}^{\downarrow X} \in \Omega_X\} = \Omega_{X \cup Y} = \{(0,0),(0,1),(1,0),(1,1)\}\} \neq c_\phi^{X \cup Y}$, contradicting equation 1.

Summarizing, in their works [8, 9] in 2011, Pouly and Kohlas make an attempt to generalize the results of Shenoy to valuation algebras equipped with a variable system, not restricting the valuations to be functions into a value set Δ. However, as proved by counterexample 1, one of the key results in their development is not correct. Since the correctness proofs provided by Pouly and Kohlas for their algorithms rely on this result, what could be a minor technical detail ends up having strong consequences for the validity of the theory as a whole.

The main objective of the next section is to identify necessary conditions for the application of the algorithms presented by Pouly and Kohlas and to prove that their correctness under those conditions.

4 Even more generic solutions in valuation algebras

During our efforts to identify the necessary conditions for the application of the algorithms presented by Pouly and Kohlas we realized that nothing in the theory we were building required that the valuation algebra was equipped with a variable system. Thus, as a byproduct of the correction effort, the resulting theory is the first one that proposes a generic algorithm, the so-called COL-LECT+EXTEND algorithm, to solve the SFP for valuation algebras which are not necessarily equipped with a variable system. The generality of the COL-LECT+EXTEND algorithm allows it to be applied to valuation algebras such as the algebra of sparse potentials, an example that until now was not covered by any previous formalization.

We start this section by generalizing the definition of the SFP problem so that it does not enforces the valuation algebra to be equipped with a variable system. Then, we introduce the concept of piecewise extensibility, and we prove

that it is a sufficient condition for the correctness of the COLLECT+EXTEND. Finally, we introduce the COLLECT+EXTENDALL algorithm, whose objective is obtaining not a single solution to the SFP, but every solution. We introduced fully piecewise extensibility and prove that it is a sufficient condition for the correctness of the COLLECT+EXTENDALL algorithm.

4.1   A more general solution finding problem

We start by introducing the concept of configuration system, a generalization of the concept of variable system that does not enforce tuples to be members of a Cartesian product. Then we generalize the SFP to configuration systems.

Configuration systems, compatibility and merge-friendliness.   We start by relating each element of the domain lattice with a set of configurations, and then we impose a minimal constraint among those sets of configurations, resulting in the notion of configuration system.

Definition 4 (Configuration system)**.** Given a lattice D, a configuration system $\langle \Gamma, \pi \rangle$ is composed of (i) a set of configurations Γ_s for each $s \in D$ and (ii) for each pair of domains $s, t \in D$ such that $s \leq t$, a surjective mapping $\pi_{t \to s} : \Gamma_t \to \Gamma_s$. Without loss of generality, the configuration sets are assumed to be mutually exclusive. Furthermore, $\Gamma_\perp = \{\lozenge\}$.

Whenever $\mathbf{x} \in \Gamma_t$, we say that $\mathbf{x}$ is a configuration with scope t. Given $\mathbf{x} \in \Gamma_t$, we note $\mathbf{x}_s = \pi_{t \to s}(\mathbf{x})$.

It is easy to see that any variable system is a configuration system. However, there are configuration systems which do not have an equivalent variable system.

A relevant concept in a configuration system is that of compatibility between configurations.

Definition 5 (Compatibility, merger, merge-friendly)**.** Let $s, t \in D$ and let $\mathbf{x} \in \Gamma_s$, and $\mathbf{y} \in \Gamma_t$. We say that $\mathbf{x}$ and $\mathbf{y}$ are *compatible* whenever there is $\mathbf{z} \in \Gamma_{s \vee t}$ such that $\mathbf{z}_s = \mathbf{x}$ and $\mathbf{z}_t = \mathbf{y}$. We say that such a $\mathbf{z}$ is a *merger* of $\mathbf{x}$ and $\mathbf{y}$. The definitions of compatibility and merger can be easily extended to a set of configurations instead of two. A configuration system is *merge-friendly* if for any $s, t \in D$, any $\mathbf{x} \in \Gamma_s$, and any $\mathbf{y} \in \Gamma_t$ whenever $\mathbf{x}_{s \wedge t} = \mathbf{y}_{s \wedge t}$, we have that $\mathbf{x}$ and $\mathbf{y}$ are compatible.

We are interested in merge-friendly configuration systems where a merger of a set of compatible configurations can be efficiently found. For example, in variable systems we can understand each tuple as restricting the values of some variables. Two tuples are compatible when there is no variable to which they assign a different value, and a merger can be easily obtained by imposing simultaneously the restrictions of both tuples.

The solution finding problem for valuation algebras with configuration systems.   The definition of solution concept can be migrated from variable

8

system to configuration system. In the latter case, a configuration system is a family $c = \{c_\phi | \phi \in \Phi\}$ of solution sets, such that for each valuation $\phi \in \Phi$, the solution set c_ϕ contains the subset of $\Gamma_{d(\phi)}$ such that $\mathbf{x} \in \Gamma_{d(\phi)}$ is considered a solution for ϕ if and only if $\mathbf{x} \in c_\phi$. The generalization of the solution finding problem is as follows

Problem 3 (SFP). Given a valuation algebra $\langle \Phi, D \rangle$ equipped with a configuration system $\langle \Gamma, \pi \rangle$, and a solution concept c, and a set of valuations $\phi_1, \ldots, \phi_m \in \Phi$, find $\mathbf{x} \in \Gamma_{d(\phi)}$ such that $\mathbf{x}$ is a solution for $\phi = \phi_1 \times \ldots \times \phi_m$.

Following Pouly and Kohlas, in order to be able to state the algorithms that solve the SFP we need the solution concept to lie inside a family of configuration extension sets. This connects the configuration system of the domain lattice with the marginalization operation of the valuation algebra and with the solution concept.

Definition 6 (Family of configuration extension sets). Given a valuation algebra $\langle \Phi, D \rangle$, and a configuration system $\langle \Gamma, \pi \rangle$ over D, and a solution concept c, a *family of configuration extension sets* is a family of sets $\mathcal{E} = \{E_\phi(\mathbf{x}) | d \in D, \phi \in \Phi, \mathbf{x} \in \Gamma_d\}$, that

1. For all $\phi \in \Phi$, for all $\mathbf{x} \in \Gamma_{d(\phi)}$,

$$E_\phi(\mathbf{x}) = \{\mathbf{x}\}. \tag{2}$$

2. For all $\phi \in \Phi$, for all $s, t \in D$ such that $s \neq t$ and $s \leq t \leq d(\phi)$, and for all $\mathbf{x} \in \Gamma_s$,

$$E_\phi(x) = \{\mathbf{y} \in \Gamma_{d(\phi)} | \mathbf{y}_t \in E_{\phi^{\downarrow t}}(\mathbf{x}) \text{ and } \mathbf{y} \in E_\phi(\mathbf{y}_t)\}. \tag{3}$$

3. For all $\phi \in \Phi$,

$$c_\phi = E_\phi(\diamond). \tag{4}$$

Whenever $\mathbf{y} \in E_\phi(x)$ we say that $\mathbf{y}$ is an extension of $\mathbf{x}$ to ϕ. Note that in order for $\mathbf{y}$ to be an extension of $\mathbf{x}$, the scope s of $\mathbf{x}$ must be smaller than the scope of $\mathbf{y}$. We can extend the definition of extension to a scope u whatsoever: given $\mathbf{x} \in \Gamma_u$ we say that $\mathbf{y} \in \Gamma_t$ is an extension of $\mathbf{x}$ to ϕ if $\mathbf{y} \in E_\phi(\mathbf{x}_{t \wedge u})$. This states that $\mathbf{y}$ is an extension of $\mathbf{x}$ if it $\mathbf{y}$ is an extension of that part of $\mathbf{x}$ which is of interest to ϕ.

Next, we introduce the COLLECT+EXTEND algorithm. The algorithm can be run on any valuation algebra equipped with a configuration system and a family of configuration extension sets. As the COLLECT algorithm, the COLLECT+EXTEND algorithm requires the existence of a covering join tree for the factorization. It has two different phases. During the first phase, the COLLECT algorithm is used to obtain the marginal of ϕ at the root of the tree. After that, during the second phase, the root starts from an empty configuration ($\diamond$), and selects a configuration that belongs to the set of extensions of $\diamond$ to his marginal. From there on, each node i of the tree receives from his parent $p(i)$ enough

---
Algorithm 2 EXTEND algorithm

Each node i of the junction tree executes
1: **if** i is the root **then**
2:     $\nu_i \leftarrow \diamond$
3: **else**
4:     From its parent $pa(i)$, receive a message ν_i.
5: **end if**
6: $\triangleright$ Extend the parent solution to i's scope.
7: Select $\eta_i \in E_{\psi'_i}(\nu_i)$.
8: **for all** j children of i **do**
9:     Send message $\nu_j = \pi_{d_i \rightarrow s(j)}(\eta_i)$ to children j.
10: **end for**
---

information from the configuration selected by $p(i)$ so that i can successfully extend it to his domain configuration set. We call this second phase the EX-TEND phase. Algorithm 2 describes it in a more precise way. At the end of the EXTEND phase, each node i of the tree has a configuration η_i over its domain. Provided that these configurations are compatible, we get a single configuration in $\Gamma_{d(\phi)}$ by assessing its merger.

4.2 Sufficient conditions for the correctness of the COL-LECT+EXTEND algorithm

In this section we consider the problem of determining under which conditions the configuration assessed by the COLLECT+EXTEND algorithm is a solution to the SFP problem. The main result is the following theorem

Theorem 3 (COLLECT+EXTEND suff. cond.). *Let $\langle \Phi, D \rangle$ be a valuation algebra equipped with a **merge-friendly** configuration system $\langle \Gamma, \phi \rangle$, and a solution concept c. Let $\mathcal{E}$ be a **piecewise extensible** family of configuration extension sets. After running the COLLECT algorithm followed by the EXTEND algorithm (Algorithms 1 and 2) over the nodes of a covering join tree $\mathcal{T}$ for $\phi = \prod_{k=1}^{n} \phi_k$, we have that there is at least a merger $\mathbf{z}$ of $\{\eta_i | i \in V\}$, and that $\mathbf{z}$ is a solution of ϕ.*

The theorem requires the family of configuration extension sets $\mathcal{E}$ to be piecewise extensible. Next, we define piecewise extensibility and then we prove that it is a sufficient condition for the correctness of the COLLECT+EXTEND algorithm.

Piecewise extensibility Intuitively, this means requiring that whenever a configuration $\mathbf{z}$ independently belongs to the set of extensions of two different valuations ϕ_1 and ϕ_2, then it does belong to the set of extensions to its product.

Definition 7. A configuration $\mathbf{x}$ (with scope s) is *extensible* to a valuation ϕ (with domain t) whenever $E_\phi(\mathbf{x}_{s \wedge t}) \neq \emptyset$. For any $\mathbf{z} \in E_\phi(\mathbf{x}_{s \wedge t})$, we say that $\mathbf{z}$ is an extension of $\mathbf{x}$ to ϕ.

10

A family of configuration extension sets $\mathcal{E}$ is *piecewise extensible* when for any two valuations $\phi_1, \phi_2 \in \Phi$ with $d_1 = d(\phi_1)$ and $d_2 = d(\phi_2)$, any $t \in D$, $d_1 \vee d_2 \geq t \geq d_1 \wedge d_2$, any $\mathbf{x} \in \Gamma_t$ and any extension $\mathbf{z}$ of $\mathbf{x}$ to both ϕ_1 and ϕ_2, we have that $\mathbf{z}$ is an extension of $\mathbf{x}$ to $\phi_1 \times \phi_2$.

Note that the piecewise extensibility requirement is defined only for pairs of valuations ϕ_1 and ϕ_2. The following lemma shows that provided that we have piecewise extensibility for two valuations, we can extend it to products of m valuations.

Lemma 1. *Let (i) $\mathcal{E}$ be a piecewise extensible family of configuration extension sets, (ii) $\phi_1, \ldots, \phi_m \in \Phi$, and $\phi = \prod_{i=1}^{m} \phi_i$, and (iii) $t \in D$, such that $\bigvee_{i=1}^{m} d_i \geq t \geq \bigvee_{i=1}^{m} r_i$, where $r_i = d_i \wedge \left(\bigvee_{j \neq i} d_j \right)$, and $d_i = d(\phi_i)$. For any $\mathbf{x} \in \Gamma_t$ and any extension $\mathbf{z}$ of $\mathbf{x}$ to $\phi_1, \ldots, \phi_m$ we have that $\mathbf{z}$ is an extension of $\mathbf{x}$ to ϕ.*

Proof. By induction on the number of terms m.

If $m = 2$, the result follows directly from the definition of piecewise extensible.

Assume it is true for $m < M$, and prove it for $m = M$. We have that $\phi = \prod_{i=1}^{M} \phi_i$. We can break it as $\phi = \phi_1 \times \prod_{i=2}^{M} \phi_i = \phi_1 \times \xi$, making $\xi = \prod_{i=2}^{M} \phi_i$.

We know $\mathbf{z}$ is an extension of $\mathbf{x}$ to ϕ_1. We can apply the induction hypothesis to $\xi = \prod_{i=2}^{M} \phi_i$, to show that $\mathbf{z}$ is also an extension of $\mathbf{x}$ to ξ and then we apply piecewise extensibility to conclude that $\mathbf{z}$ is an extension of $\mathbf{x}$ to ϕ.

In order to apply the induction hypothesis, we take $\xi = \prod_{i=2}^{M} \phi_i$, and $t' = t \wedge d(\xi)$. It is easy to see that $\bigvee_{i=2}^{M} d_i = d(\xi) \geq t \wedge d(\xi) = t'$. On the other hand, $t' \geq \bigvee_{i=2}^{M} r_i$. Since $\mathbf{z}$ is an extension of $\mathbf{x}_{t'}$ to $\phi_2, \ldots, \phi_M$, we have that for each i, $\mathbf{z} \in E_{\phi_i}(\mathbf{x}_{s_i})$, where $s_i = t \wedge d_i$.

Note that $d(\xi) = \bigvee_{i=2}^{M} d_i$. Take $t' = t \wedge d(\xi)$. We can see that $\bigvee_{i=2}^{M} d_i \geq t' \geq \bigvee_{i=2}^{M} r_i$. Furthermore, for $i \neq 1$, we have that $s_i' = t' \wedge d_i = t \wedge d(\xi) \wedge d_i = t \wedge (\bigvee_{i=2}^{M} d_i) \wedge d_i = t \wedge d_i = s_i$, and hence $\mathbf{z}$ is an extension of $\mathbf{z} \in E_{\phi_i}(\mathbf{x}_{s_i'})$. Applying the induction hypothesis we have that $\mathbf{z}_{d(\xi)} \in E_\xi(\mathbf{x}_{t'}) = E_\xi(\mathbf{x}_{t \wedge d(\xi)})$.

The conditions to apply piecewise extensibility to $\phi_1 \times \xi$ are now in place. Observe that $d_1 \vee d(\xi) \geq t \geq \left(\bigvee_{i=2}^{M} d_i \right) \wedge d_1 = d(\xi) \wedge d_1$, that $\mathbf{z}_{d_1} \in E_{\phi_1}(\mathbf{x}_{s_1})$, and that $\mathbf{z}_{d(\xi)} \in E_\xi(\mathbf{x}_{t \wedge d(\xi)})$. Since $\mathcal{E}$ is piecewise extensible we have that $\mathbf{z}$ is an extension of $\mathbf{x}$ to ϕ. □

Sufficient condition for the correctness of COLLECT+EXTEND. Next, we see that on piecewise extensible family of configuration extension sets, it is possible to take benefit of the factorization of the valuation to find a solution by merging partial solutions to the different factors which are coherent between them. We start by proving this for a product of two valuations and a product of m valuations. Then we apply those results to prove that the COLLECT+EXTEND algorithm is correct.

We start proving the following lemma, that shows that provided we have piecewise extensibility, for any valuation that is the product of two factors, if

we are given a solution to the projection of the product to the domain of one of the factors and an extension of that solution to the second factor, the merger of these two is a solution to the product.

Lemma 2. *Let $\mathcal{E}$ be a piecewise extensible family of configuration extension sets. Let $\phi_1, \phi_2 \in \Phi$, and let $d_1 = d(\phi_1)$, $d_2 = d(\phi_2)$, and $\phi = \phi_1 \times \phi_2$. For any $\mathbf{x} \in c_{\phi \downarrow d_1}$, any extension $\mathbf{y}$ of $\mathbf{x}$ to ϕ_2, and any merger $\mathbf{z}$ of $\mathbf{x}$ and $\mathbf{y}$, we have that $\mathbf{z} \in c_\phi$.*

Proof. First, we will use piecewise extensibility to prove that $\mathbf{z} \in E_\phi(\mathbf{x})$. To do that we apply definition 7 with $t = d_1$. By hypothesis, we have that $\mathbf{z}_{d_2} = \mathbf{y} \in E_{\phi_2}(\mathbf{x}_{d_2 \wedge d_1})$. From equation 2, we have that $E_{\phi_1}(\mathbf{x}) = \{\mathbf{x}\}$. Hence, $\mathbf{z}_{d_1} = \mathbf{x} \in E_{\phi_1}(\mathbf{x})$. Thus, $\mathbf{z}$ is a coherent extension of $\mathbf{x}$ to both ϕ_1 and ϕ_2 and we can apply piecewise extensibility to conclude that $\mathbf{z} \in E_\phi(\mathbf{x})$.

Then, we can jointly apply equations 4 and 3 to conclude that $\mathbf{z} \in c_\phi = E_\phi(\diamond)$. □

As in the previous section, we can generalize this result to products of m valuations.

Lemma 3. *Let $\mathcal{E}$ be a piecewise extensible family of configuration extension sets. Let $\phi = \phi_\rho \times \prod_{i=1}^m \phi_i$, with $d_i = d(\phi_i)$, $d_\rho = d(\phi_\rho)$, $r_i = d_i \wedge \left(\bigvee_{j \neq i} d_j \right)$, and $d_\rho \geq \bigvee_{i=1}^m r_i$. Given $\mathbf{x} \in c_{\phi \downarrow d_\rho}$, for each $1 \leq i \leq m$, $\mathbf{y}_i$ an extension of $\mathbf{x}$ to ϕ_i, and any merger $\mathbf{z}$ of $\mathbf{x}$ and $\mathbf{y}_1, \ldots, \mathbf{y}_m$, we have that $\mathbf{z} \in c_\phi$.*

Proof. Let By induction on m.

If $m = 1$, we can directly apply Lemma 2.

Assume it is true for $m < M$. We have to prove that it is true for $m = M$. Let $\xi = \prod_{i=1}^m \phi_i$. First, we will prove that $\mathbf{z}_{d_\xi} \in E_\xi(\mathbf{x}_{d_\rho \wedge d_\xi})$ and then we will apply Lemma 2. To see that $\mathbf{z}_{d_\xi} \in E_\xi(\mathbf{x}_{d_\rho \wedge d_\xi})$, we apply Lemma 1 with $t = d_\rho \wedge d_\xi$.

We need to verify that $\bigvee_{i=1}^m d_i \geq t \geq \bigvee_{i=1}^m r_i$. The left inequality is satisfied since $t = d_\rho \wedge d_\xi = d_\rho \wedge (\bigvee_{i=1}^m d_i) \leq \bigvee_{i=1}^m d_i$. Since for all i, $r_i \leq d_\xi$, we have that $\bigvee_{i=1}^m r_i \leq d_\xi$. Since by hypothesis we have that $d_\rho \geq \bigvee_{i=1}^m r_i$, we can conclude that $d_\rho \wedge d_\xi \geq \bigvee_{i=1}^m r_i$.

Furthermore we can verify that for each $i \in \{1, \ldots, m\}$, $\mathbf{z}_{d_i} = \mathbf{y}_i \in E_{\phi_i}(\mathbf{x}_{t \wedge d_i})$. Note that $t \wedge d_i = d_\rho \wedge d_\xi \wedge d_i = d_\rho \wedge (\bigvee_{i=1}^m d_i) \wedge d_i = d_\rho \wedge d_i$, and by hypothesis we have that $\mathbf{z}_{d_i} = \mathbf{y}_i \in E_{\phi_i}(\mathbf{x}_{d_\rho \wedge d_i},)$.

Applying Lemma 1, we get that $\mathbf{z}_{d_\xi} \in E_\xi(\mathbf{x}_{d_\rho \wedge d_\xi})$ and since by hypothesis we have that $\mathbf{x} \in c_{\phi \downarrow d_\rho}$, we can conclude from Lemma 2 that $\mathbf{z} \in c_{\phi_\rho \times \phi_\xi} = c_\phi$. □

The former results allow us to state the following theorem (the main result of the section) proving that when a valuation breaks as a product of smaller valuations, the COLLECT+EXTEND algorithm can be used to assess a solution to it.

Theorem 3 (COLLECT+EXTEND suff. cond.)**.** *Let $\langle \Phi, D \rangle$ be a valuation algebra equipped with a **merge-friendly** configuration system $\langle \Gamma, \phi \rangle$, and a solution concept c. Let $\mathcal{E}$ be a **piecewise extensible** family of configuration extension*

sets. After running the COLLECT *algorithm followed by the* EXTEND *algorithm (Algorithms 1 and 2) over the nodes of a covering join tree $\mathcal{T}$ for $\phi = \prod_{k=1}^{n} \phi_k$, we have that there is at least a merger $\mathbf{z}$ of $\{\eta_i | i \in V\}$, and that $\mathbf{z}$ is a solution of ϕ.*

Proof. By induction on the number of nodes of the junction tree. If the junction tree has only one node then the proof is trivial.

Assume that the junction tree has $m > 1$ nodes. We can see that the conditions to apply Lemma 3 are satisfied at the root.

1. $\phi = \psi_\rho \times \prod_{i \in Children(\rho)} \xi_i$, where $\xi_i = \prod_{j \in \mathcal{T}_i} \psi_i$. Let $d'_i = d(\xi_i)$

2. For each $i, j \in Children(\rho)$ such that $i \neq j$, due to the running intersection property we have that $d_\rho \geq d'_i \wedge \left(\bigvee_{j \neq i} d'_j \right)$.

3. $\mathbf{x} \in c_{\phi \downarrow d_\rho}$.

4. For each $i \in Children(\rho)$, $\mathbf{y}_i \in E_{\xi_i}(\mathbf{x}_{d_\rho \wedge d'_i})$

5. $\mathbf{z}$ is a merger of $\mathbf{x}$ and each of the $\mathbf{y}'_i s$

As a consequence of lemma 3, we can conclude that $\mathbf{z} \in c_\phi$. $\qquad\square$

Fully piecewise extensibility and assessing all solutions. By strengthening the concept of piecewise extensibility, we can use an algorithm similar to COLLECT+EXTEND to assess all solutions instead of only one. The strengthening is named fully piecewise extensibility.

Definition 8. A family of configuration extension sets $\mathcal{E}$ is *fully piecewise extensible* when for any two valuations $\phi_1, \phi_2 \in \Phi$ with $d_1 = d(\phi_1)$ and $d_2 = d(\phi_2)$, any $t \in D$, $d_1 \vee d_2 \geq t \geq d_1 \wedge d_2$, any $\mathbf{x} \in \Gamma_t$ and any $\mathbf{z}$, we have that $\mathbf{z}$ is an extension of $\mathbf{x}$ to both ϕ_1 and ϕ_2 , **if and only if** $\mathbf{z}$ is an extension of $\mathbf{x}$ to $\phi_1 \times \phi_2$.

The algorithm and the theorem that shows that this is a sufficient condition are omitted due to lack of space but can be derived without effort.

5   Conclusions

We have corrected and generalized the theory and algorithms for the generic construction of solutions in valuation based systems. To the best of our knowledge, these results provide the most general theory for dynamic programming up-to-date, covering commonly used examples such as finding the maximum of a combination of sparse functions, which the current theory did not cover.

References

[1] R.E. Bellman. *Dynamic Programming*. Princeton University Press, 1957.

[2] Umberto Bertelè and Francesco Brioschi. *Nonserial Dynamic Programming*, volume 91 of *Mathematics in Science and Engineering*. Academic Press, 1972.

[3] B.A. Davey and H.A. Priestley. *Introduction to lattices and order*. Cambridge University Press, second edition, 2008.

[4] George Grätzer. *Lattice Theory: Foundation*. Springer Basel, 2011.

[5] J Kohlas and N Wilson. Semiring induced valuation algebras: Exact and approximate local computation algorithms. *Artificial Intelligence*, 172(11):1360–1399, July 2008.

[6] L. G. Mitten. Composition Principles for Synthesis of Optimal Multistage Processes. *Operations Research*, 12(4):610–619, 1964.

[7] George L. Nemhauser. *Introduction to Dynamic Programming*. John Wiley & Sons, 1966.

[8] Marc Pouly. Generic solution construction in valuation-based systems. *Advances in Artificial Intelligence*, pages 335–346, 2011.

[9] Marc Pouly and Jürg Kohlas. *Generic Inference*. John Wiley & Sons, Hoboken, NJ, USA, May 2011.

[10] Prakash P Shenoy. Axioms for Dynamic Programming. In A. Gammerman, editor, *Computational Learning and Probabilistic Reasoning*, pages 259–275. John Wiley & Sons, Ltd., 1996.