Minimizing Communication Costs of Distributed Local Computation[1]

Pouly Marc[2]

Abstract. From the very first beginning, local computation algorithms were considered as distributed algorithms wherein (virtual) processors exchange messages to solve a common computational task. Thereby, the efficiency of local computation is particularly pointed out, whenever the complexity of the underlying valuation algebra operations manifests an exponential complexity. This behavior is often related to the size of the involved messages and therefore, minimization of communication costs in local computation becomes an important issue. This paper measures these costs by use of a generic weight function and shows concretely, under which condition these costs can be minimized. Furthermore, we present an efficient algorithm for the appropriate minimization problem.

1 INTRODUCTION

Over the last few decades, a lot of research was dedicated to fathom the definition and essence of information. A particular approach towards these objectives is known as a valuation algebra which has been introduced in [13, 12] and studied in more detail by [4]. This mathematical framework defines knowledge and information by their principal operations of combination and marginalization, which realize aggregation and extraction of knowledge respectively. Based on this system, a set of efficient algorithms to process this representation of knowledge were devised. They are pooled under the name local computation and among them, there exists different architectures which exploit the particular properties of valuation algebra instances to augment the computation's efficiency.

A further important property is that information and knowledge are distributed resources. Information may come from different sources, is shared and processed in a distributed manner. All these actions require that pieces of information can be transmitted between the hosts of a common network, as it is done virtually by the message passing in local computation. If we apply these techniques on a real processor network, new questions arise that mainly concern the allocation of processors and the impact of communication costs caused by the local computation messages. The importance of the latter is especially invigorated by the fact that the operations of most valuation algebra instances adopt an exponential behavior which normally carries over to the message sizes in local computation.

[1] Research supported by grant No. 20-67996.02 of the Swiss National Foundation for Research.

[2] University of Fribourg, Switzerland, email: marc.pouly@unifr.ch

The aim of this paper is thus to investigate the application of local computation techniques on physically distributed knowledgebases. We first complement the valuation framework in such a way, that communication costs of transmitting valuations can be modeled. Then, based on results from [11], we show how the nodes in the underlying tree structure of local computation can be associated with real-world processors. This is a fair realization of classical local computation philosophy, were join tree nodes are regarded as virtual processors [13, 4] and local computation messages are exchanged between them. Our main interest is directed towards the total communication costs caused by these messages and naturally, we will investigate in which situations one can minimize communication costs in local computation.

2 VALUATION ALGEBRAS

The basic elements of a valuation algebra are so-called *valuations*. Intuitively, a valuation can be regarded as a representation of knowledge about the possible values of a set of variables. It can be said that each valuation ϕ refers to a finite set of variables $d(\phi)$, called its *domain*. For an arbitrary set s of variables, Φ_s denotes the set of valuations ϕ with $d(\phi) = s$. With this notation, the set of all possible valuations corresponding to a finite set of variables r can be defined as

$$\Phi = \bigcup_{s \subseteq r} \Phi_s.$$

Let D be the lattice of subsets (the powerset) of r. For a single variable X, Ω_X denotes the set of all its possible values. We call Ω_X the *frame* of the variable X. In an analogous way, we define the frame of a non-empty variable set $s \in D$ by the Cartesian product of frames Ω_X of each variable $X \in s$,

$$\Omega_s = \prod_{X \in s} \Omega_X. \tag{1}$$

The elements of Ω_s are called *configurations* of s. The frame of the empty variable set is defined by convention as $\Omega_\emptyset = \{\diamond\}$. This summarizes the most important notions that are used to define a *valuation algebra* formally.

2.1 A Formal Definition

Let Φ be a set of valuations with their domains in D. We assume the following operations defined on Φ and D:

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

- *Combination:* $\Phi \times \Phi \to \Phi$; $(\phi, \psi) \mapsto \phi \otimes \psi$,
- *Marginalization:* $\Phi \times D \to \Phi$; $(\phi, x) \mapsto \phi^{\downarrow x}$, for $x \subseteq d(\phi)$.

These are the three basic operations of a valuation algebra. We impose now the following set of axioms on Φ and D:

1. *Commutative Semigroup:* Φ is associative and commutative under combination.
2. *Labeling:* For ϕ, $\psi \in \Phi$,
$$d(\phi \otimes \psi) = d(\phi) \cup d(\psi).$$
3. *Marginalization:* For $\phi \in \Phi$, $x \in D$, $x \subseteq d(\phi)$,
$$d(\phi^{\downarrow x}) = x.$$
4. *Transitivity:* For $\phi \in \Phi$ and $x \subseteq y \subseteq d(\phi)$,
$$(\phi^{\downarrow y})^{\downarrow x} = \phi^{\downarrow x}.$$
5. *Combination:* For ϕ, $\psi \in \Phi$ with $d(\phi) = x$, $d(\psi) = y$ and $z \in D$ such that $x \subseteq z \subseteq x \cup y$,
$$(\phi \otimes \psi)^{\downarrow z} = \phi \otimes \psi^{\downarrow z \cap y};$$
6. *Domain:* For $\phi \in \Phi$ with $d(\phi) = x$,
$$\phi^{\downarrow x} = \phi.$$

The axioms require natural properties regarding knowledge or information modelling. The first axiom indicates that Φ is a commutative semigroup under combination. If information comes in pieces, the sequence does not influence the overall knowledge. The labeling axiom tells us that the combination of valuations gives knowledge over the union of the involved domains. Neither variables vanish, nor new appear. The marginalization axiom expresses the natural functioning of focusing. Transitivity says that marginalization can be performed in several steps. The combination axiom tells us that we either combine a new piece to the already given knowledge and focus afterwards to the desired domain, or we cut first the uninteresting parts of the new knowledge out and combine it afterwards. There is no difference between the two approaches. Finally, the domain axiom tells us that knowledge is not influenced by projecting it to the own domain. Without this axiom, this is not always the case [12].

Definition 1 *A system (Φ, D) together with the operations of labeling, marginalization and combination satisfying these axioms is called a* valuation algebra.

Based on this definition, there are many variations being either more or less restrictive. We enumerate consecutively the most important ones and refer to [4] and [11] for more details.

- *Valuation algebras with neutral elements:* Every subsemigroup Φ_s has a neutral element, i.e. for all $s \in D$ there is an element $e_s \in \Phi_s$ such that for all $\phi \in \Phi$ with $d(\phi) = s$ we get $\phi \otimes e_s = \phi$. For consistency, we add a neutrality axiom to the above system:

  7. *Neutrality:* For $x, y \in D$,
  $$e_x \otimes e_y = e_{x \cup y}.$$

- *Separative and regular valuation algebras:* In separative and regular valuation algebras several conditions are required on (Φ, D) such that every valuation $\phi \in \Phi$ has a kind of inverse ϕ^{-1}. In the first, the newly required properties allow for an embedding of (Φ, D) into a valuation algebra (Φ^*, D) with a well-defined division operator possible. Unfortunately, the marginalization operation in (Φ^*, D) is then only partial in general. This system is fully described in [11]. The regular kind of division requires stronger conditions, such that the embedding with the drawback of the partial marginals can be avoided.
- *Information algebras:* An information algebra is a valuation algebra with neutral elements and the following additional axioms

  8. *Idempotency:* For all $\phi \in \Phi$ and $x \subseteq d(\phi)$,
  $$\phi \otimes \phi^{\downarrow x} = \phi.$$

  9. *Stability:* For all $\phi \in \Phi$ and $x \subseteq y$,
  $$e_y^{\downarrow x} = e_x.$$

  Axiom 8 states that a valuation combined with a part of itself gives nothing new. This is a very natural property of information and is therefore eponym for this variation of a valuation algebra. Among the many strong consequences of this definition, which are altogether listed in [4], it permits to introduce a partial ordering relation of valuations. For an information algebra (Φ, D) and $\phi, \psi \in \Phi$, we say that $\phi \geq \psi$ if and only if $\phi \otimes \psi = \phi$.

An important point is that in the above definition, the existence of neutral elements is not mandatory for a valuation algebra. This is justified by the fact, that we can always adjoin a so-called *identity element* e to a valuation algebra, whenever no neutral elements are available. The consequence is that the definition of a valuation algebra becomes more general. It covers instances which do not provide neutral elements (see [4] for an example) and avoids problems that occur when neutral elements are not explicitly representable. Furthermore, local computation is generally more efficient if we use identity elements instead of neutral elements to initialize join tree nodes. We refer to [11] for a complete discussion of these consequences and focus again on the formal introduction of identity elements.

Let (Φ, D) be an arbitrary valuation algebra. We add a new valuation e to Φ and denote the resulting system by (Φ', D). The operations are extended from Φ to Φ' in the following way:

1. *Labeling:* $\Phi' \to D$; $\phi \mapsto d'(\phi)$,
   - $d'(\phi) = d(\phi)$, if $\phi \in \Phi$;
   - $d'(e) = \emptyset$;
2. *Combination:* $\Phi' \times \Phi' \to \Phi'$; $(\phi, \psi) \mapsto \phi \otimes' \psi$,
   - $\phi \otimes' \psi = \phi \otimes \psi$ if $\phi, \psi \in \Phi$;
   - $\phi \otimes' e = e \otimes' \phi = \phi$ if $\phi \in \Phi$;
   - $e \otimes' e = e$;
3. *Marginalization:* $\Phi' \times D \to \Phi'$; $(\phi, x) \mapsto \phi^{\downarrow' x}$, for $x \subseteq d(\phi)$
   - $\phi^{\downarrow' x} = \phi^{\downarrow x}$ if $\phi \in \Phi$;

- $e^{\downarrow' \emptyset} = e$.

Theorem 1 *(Φ', D) with the extended operations d', $\otimes'$ and $\downarrow'$ is a valuation algebra. If (Φ, D) already provides neutral elements, then there is a valuation $e_\emptyset \in \Phi$ having the same properties in (Φ, D) as e in (Φ', D).*

A proof of this theorem can be found in [11], which shows also that all additional properties listed above are carried over to the extension (Φ', D). We usually identify the operators in (Φ', D) with those in (Φ, D) if they are not used to distinguish between the two algebras.

2.2 Weight Functions

For out future studies of communication costs in context of local computation, we introduce a generic weight function for valuation algebras that, intuitively, measures the amount of memory used to store a given valuation.

Definition 2 *Let (Φ, D) be a valuation algebra. A function $\omega : \Phi \to \mathbb{N}_0$ is called weight function if for all $\phi \in \Phi$ and $x \subseteq d(\phi)$ we have $\omega(\phi) \geq \omega(\phi^{\downarrow x})$.*

This definition can be extended to a valuation algebras Φ^* with an identity element e by assigning a constant value to $\omega(e)$. Because the identity element e has been adjoint artificially and has no informational content, it is reasonable to set $\omega(e) = 0$.

Essentially, the above definition states that the weight of a valuation decreases when it is projected to a subdomain. This seems very natural in the sense that $\phi^{\downarrow x}$ is contained in ϕ and therefore contains somehow less information which carries over to its memory requirements. However, there are instances which contradict this intuition. This is typically the case when valuation algebra instances are defined on language level rather than on models [4, 8]. A well-known example can be found in propositional logic [5, 7]. Defining valuations as being sets of clauses can cause an increase of the number of clauses when the marginalization operator is applied. But on the other hand, when the model-based dual representation of valuations is chosen, the above requirement is fulfilled.

Of special interest are so-called *weight-predictable* valuation algebras whose weight function only depends on the valuation's domain. This is expressed by the following definition.

Definition 3 *Let ω be a weight function for a valuation algebra (Φ, D). ω is called a weight predictor for Φ if there exists a function $f : D \to \mathbb{N}_0$ such that for all $\phi \in \Phi$*

$$\omega(\phi) = f(d(\phi)).$$

Especially natural is the weight predictability property in the case of semiring induced valuation algebras [6] which are defined over a complete set of configurations. But there are also numerous other instances that adopt this property such as for example Gaussian potentials [4].

3 COMMUNICATION COST MODEL

Let $P = \{p_1, p_2, \ldots, p_p\}$ be a set of processors participating in a communication network. Processors are considered as independent computing units with their own private memory space such that they can execute tasks without interactions and in parallel. In order to model the costs of transmitting valuations between processors, we need to define a cost function that satisfies a number of natural requirements. Let $c_\phi(i, j)$ denote the communication costs of sending valuation ϕ from processor i to processor j, presuming that ϕ is already stored on processor i. We impose the following requirements on c:

- $c_\phi(i, j) \geq 0$,
- $c_\phi(i, j) = c_\phi(j, i)$,
- $i = j \;\Rightarrow\; c_\phi(i, j) = 0$.

For a more concrete model, we assume that the distance between each pair of processors i and j in the underlying network topology is given in a distance matrix $D = [d_{i,j}]$. It is natural to assume $d_{i,i} = 0$ and $d_{i,j} = d_{j,i}$. Then, our cost model to transmit valuations between processors is defined as follows: Let (Φ, D) be a valuation algebra with a weight function ω. For all $\phi \in \Phi$ and $i, j \in P$, we define

$$c_\phi(i, j) \quad = \quad \omega(\phi) \cdot d_{i,j}. \tag{2}$$

Clearly, this definition satisfies the above requirements for communication cost functions. It allows to represent incomplete network topologies by defining $d_{i,j} = \infty$, if no communication channel between the processors i and j exists.

3.1 Costs of Local Computation

We introduced valuation algebras as a suitable framework to represent knowledge such that we can now focus on the treatment of this knowledge. In the terms of valuation algebras, this task is expressed by the *projection problem* that consists in combining a multiset of valuations and projecting the result onto the domains of interest. The distribution of knowledge comes into play as soon as projection problems whose valuations reside on different processors are considered. This is expressed by the following extension:

Definition 4 *A distributed projection problem for a set of queries $\{x_1, \ldots, x_s\}$, $x_i \subseteq d(\phi_1) \cup \ldots \cup d(\phi_n)$, is given by the computational tasks*

$$(\phi_1 \otimes \phi_2 \otimes \ldots \otimes \phi_n)^{\downarrow x_i}$$

together with an assignment mapping $\chi : \{\phi_1, \ldots, \phi_n\} \to P$ determining the host processor of each valuation ϕ_i.

Executing such a sequence of combinations, followed by a single marginalization turns out to be computationally infeasible, because the complexity of the operations tends to increase exponentially with the size of the involved domains. Local computation goes around this problem by offering a set of efficient algorithms that schedule the operations in such a way, that the domains do not grow significantly. These algorithms are described as message passing schemes on graphical structures called join trees.

Definition 5 *A join tree is a labeled tree (V, E, λ) whose labeling function $\lambda : V \to D$ satisfies the running intersection property, i.e. for two nodes $v_1, v_2 \in V$, if $X \in \lambda(v_1) \cap \lambda(v_2)$, then X is contained in every node label on the unique path between v_1 and v_2.*

[11] describes in detail the way from a given projection problem towards a suitable join tree. In doing so, the important objective is that the join tree covers both, the domains of the factors in the projection problem, often called knowledgebase factors, and the queries. This is the case, if the join tree has the following properties:

- for each factor ϕ_i, there is a node $v \in V$ with $d(\phi_i) \subseteq \lambda(v)$,
- for each query x_i, there is a node $v \in V$ with $x_i \subseteq \lambda(v)$.

Such a join tree is called *covering join tree*.

The next step consists in distributing the knowledgebase factors among the nodes. This is done by an assignment mapping $a : \{1, 2, \ldots, n\} \to V$, that assigns to every factor ϕ_i a join tree node $a(i) \in V$ with $d(\phi_i) \subseteq \lambda(a(i))$. If multiple factors are assigned to the same node, they need to be combined which would cause antecedent communication costs. Therefore, we will add to those nodes sufficiently many new parents in order to obtain a join tree with at most one knowledgebase factor per node. This results in a new factorization

$$\phi_1 \otimes \ldots \otimes \phi_n \quad = \quad \psi_1 \otimes \ldots \otimes \psi_m, \tag{3}$$

where $m = |V|$ and

$$\psi_i \quad = \quad \begin{cases} \phi_j & \text{if } a(j) = i, \\ e & \text{otherwise.} \end{cases} \tag{4}$$

ψ_i denotes the content of node i, which corresponds either to one of the knowledgebase factors ϕ_j or to the identity element e. This induces naturally a new processor assignment mapping $\xi : V \to P$ which is defined for all $i \in V$ as:

$$\xi(i) \quad = \quad \begin{cases} \chi(\phi_j) & \text{if } \psi_i = \phi_j, \\ p \in P \text{ arbitrary} & \text{otherwise.} \end{cases} \tag{5}$$

To each node, we assign the processor where its factor is hosted. Because the identity elements have been introduced artificially, an arbitrary processor is assigned to these nodes. This will serve primarily to estimate the total communication costs of local computation and we refer to section 4 that shows how a more sophisticated assignment will minimize these costs. Our identification of join tree nodes and processors colludes stainlessly with classical theory of local computation, where join tree nodes are often regarded as virtual processors, exchanging messages with their neighboring nodes.

In the Shenoy-Shafer architecture, we assume the existence of *mailboxes* between each pair of linked nodes in the join tree. These mailboxes serve as message caches. Once a message is sent from one node to its neighbor, it is stored in the interjacent mailbox and stays available up to the end of the algorithm. The message itself, sent from node i to node j, is defined as:

$$\mu_{i \to j} = \left(\psi_i \otimes \bigotimes_{k \in ne(i), j \neq k} \mu_{k \to i} \right)^{\downarrow \sigma_{i \to j} \cap \lambda(j)}, \tag{6}$$

with

$$\sigma_{i \to j} = d(\psi_i) \cup \bigcup_{k \in ne(i), j \neq k} d(\mu_{k \to i}). \tag{7}$$

$ne(i)$ denotes the set of neighbors of node i, i.e. all nodes that are directly linked with node i in the join tree. Each node then combines its initial node content with all messages received from the neighbors and the result of these computations is stated by the following theorem.

Theorem 2 *At the end of the message passing in the Shenoy-Shafer architecture, we obtain at node i*

$$\phi^{\downarrow \lambda(i)} = \psi_i \otimes \bigotimes_{j \in ne(i)} \mu_{j \to i}. \tag{8}$$

The proof of this theorem can be found in [11]. It is pointed out that we can always find a possible scheduling of this algorithm by starting from the leaf nodes which do not have any parents and therefore are able to compute their messages according to formula 6.

Answering the projection problem from the result demands one last marginalization per query x_i,

$$\phi^{\downarrow x_i} = \left(\phi^{\downarrow \lambda(i)} \right)^{\downarrow x_i}.$$

By use of the weight predictable cost model, we obtain the following upper bound for the communication costs in the Shenoy-Shafer architecture:

$$\begin{aligned} T_\xi \quad &= \quad \sum_{i=1}^{m} \sum_{j \in ne(i)} f(\sigma_{i \to j} \cap \lambda(j)) \cdot d_{\xi(i), \xi(j)} \\ &\leq \quad \sum_{i=1}^{m} \sum_{j \in ne(i)} f(\lambda(i) \cap \lambda(j)) \cdot d_{\xi(i), \xi(j)} \\ &= \quad 2 \sum_{i=1}^{m-1} f(\lambda(i) \cap \lambda(ch(i))) \cdot d_{\xi(i), \xi(ch(i))}, \end{aligned}$$

where $ch(i)$ denotes the unique neighbor of node i towards the root, i.e. its child node. The last equality holds because there are always two messages sent along every edge in the join tree.

Executing local computation on a weight predictable valuation algebra represents in a way the situation of full information. We can predict the communication costs of each message that will be sent during the algorithm's run without effectively computing this message.

4 MINIMIZATION PROCESS

In the foregoing section we constructed a join tree factorization

$$\phi_1 \otimes \ldots \otimes \phi_n \quad = \quad \psi_1 \otimes \ldots \otimes \psi_m,$$

from a given projection problem in such a way that each factor ψ_i corresponds either to a knowledgebase factor ϕ_j or to the identity element e. We bring out this important partition as follows:

$$\Psi = \{\psi_1, \psi_2 \ldots \psi_m\} = \Psi_{\overline{e}} \cup \Psi_e,$$

where $\Psi_e = \{\psi \in \Psi : \psi = e\}$ and $\Psi_{\overline{e}} = \Psi \setminus \Psi_e$.

In order to minimize the total communication costs, we are looking for a processor assignment mapping ξ that minimizes

T_ξ, respectively its upper bound. More concretely, we need to find a processor assignment mapping ξ, such that

$$\sum_{i=1}^{m-1} f_i \cdot d_{\xi(i),\xi(ch(i))} \quad \leq \quad B \tag{9}$$

for a given upper bound $B \in \mathbb{N}$ and $f_i = f(\lambda(i) \cap \lambda(ch(i)))$. There are $|P|^{|\Psi_e|}$ possible assignment mappings ξ because we can only vary the processor assignment of those nodes that initially hold an identity element. Clearly, a brute force determination becomes infeasible with an increasing cardinality of Ψ_e. Subsequently, we will refer to this decision problem as the *partial distribution problem* (*PDP*), due to its task of completing a partial processor distribution over join tree nodes.

4.1 Analysis of PDP

[1] originally initiated the study of the *multiway cut problem* whose close relation to PDP is the topic of this section. For our purposes, we stress the more general and illustrative definition of multiway cut given in [2]:

Instance: Given a simple graph $G = (V, E)$, a set of colors $C = \{1, 2, \ldots, r\}$, a positive number $B \in \mathbb{N}$ and a weight function $w : E \to \mathbb{N}$, assigning a weight $w(i, j)$ to each edge $(i, j) \in E$. Furthermore, we assume for $N \subseteq V$ a given partial coloration $\nu : N \to C$.

Question: Is there a completed mapping $\overline{\nu} : V \to C$, such that $\overline{\nu}(i) = \nu(i)$, if $i \in N$ and

$$\sum_{(i,j)\in E,\ \overline{\nu}(i)\neq\overline{\nu}(j)} w(i, j) \quad \leq \quad B.$$

This definition includes the so-called *color-independent* version of a weight function, which has also been used in [1]. A more general form is proposed by [2] and defined as $w : E \times C \times C \to \mathbb{N}$. In this case, the weight function is called *color-dependent* and the number $w(i, j, p, q)$ specifies the weight of the edge $(i, j) \in E$, if $\overline{\nu}(i) = p$ and $\overline{\nu}(j) = q$. Clearly, color-independence is reached, if for any $(i, j) \in E$, $p_1 \neq q_1$ and $p_2 \neq q_2$, we have $w(i, j, p_1, q_1) = w(i, j, p_2, q_2)$. Finally, if $w(i, j) = c$ for all $(i, j) \in E$, the weight function is said to be *constant*. Note, that without loss of generality, we can assume $c = 1$.

Figure 1 shows a color-independent multiway cut instance with three colors, together with a possible coloration of total weight 28.

[1] pointed out that the multiway cut problem is NP-complete even for $|N| = 3$, $|N_i| = 1$ and constant weight functions. But for the special case of a tree, [2] showed that multiway cut can be solved in polynomial time even for color-dependent weight functions. The corresponding algorithm has time complexity $O(|V| \cdot r^2)$. This finally determines the complexity of the partial distribution problem, associated with the minimization of communication costs in local computation with weight predictable valuation algebras.

Theorem 3 *PDP is solvable in polynomial time.*

Proof. Assume a given PDP instance on a join tree $G = (V, E)$. We define $N = \Psi_{\overline{e}}$ and $C = \{c_1, c_2 \ldots, c_{|P|}\}$.

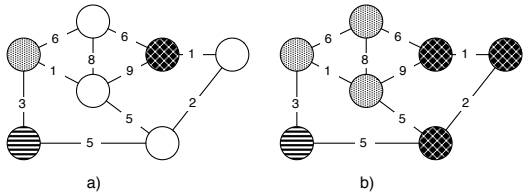

Figure 1. a) A color-dependent multiway cut instance with three colors $C = \{dotted, dashed, grilled\}$. The numbers labeling the graph's edges represent the weight function w. b) A possible coloration with total weight 28.

The weights of the multiway cut instance are given by $w(i, j, p, q) = f_i \cdot d_{p,q}$ for $j = ch(i)$ and the initial coloration of the multiterminal cut instance $\nu : N \to C$ is defined as: $\nu(i) = c_j$ if $\chi(\psi_i) = p_j$ for $i \in N$ and χ being the processor assignment mapping of the distributed projection problem. The upper bound B is equal for both instances.

Let $\overline{\nu}$ be a solution of the constructed multiway cut problem. Clearly, this implies a solution ξ of the corresponding PDP instance by defining: $\xi(i) = p_j$ if $\overline{\nu}(i) = c_j$. Therefore, and because G is a tree, we know that solving PDP has polynomial time complexity. □

This result shows that local computation is not only an efficient way to compute projection problems but the underlying join tree structure ensures also that in the case of weight predictable valuation algebras, communication costs caused by distributed projection problems can indeed be minimized with reasonable effort. A corresponding algorithm to solve the partial distribution problem efficiently is the objective of the following subsection.

4.2 Minimization Algorithm

In the proof of theorem 3 we have seen that PDP basically reduces to an application of the color-dependent multiway cut problem. Because of this tight relation, we will now reformulate the polynomial algorithm for the multiway cut problem given in [2] in terms of the PDP, such that it can be applied directly to the problem at hand. In this way, [10] showed that it is sufficient to consider only PDP instances with the property that only leaf nodes initially contain a knowledgebase factor and that identity elements are assigned to all other nodes, i.e. $\{i \in V : \psi_i \neq e\} = le(V)$ with $le(V) = \{i \in V : pa(i) = \emptyset\}$. An example of such an instance s shown in figure 2.

The processor assignment algorithm that minimizes communication costs can now be formulated as a two-phase process. The first phase corresponds to an inward tree propagation which assigns penalty values to each node. Afterwards, the second phase propagates outwards and assigns a processor to all interior nodes such that the penalty values are minimized.

Phase I: For each leaf $v \in le(V)$, we define:

$$pen_i(v) = \begin{cases} 0 & \text{if } \chi(\psi_v) = i, \\ \infty & \text{otherwise.} \end{cases}$$

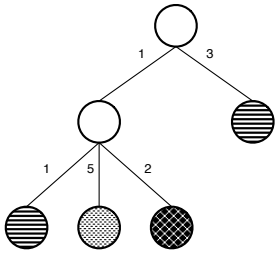

Figure 2. A PDP instance with three processors $P = \{dotted,\ dashed,\ grilled\}$ satisfying the imposed restriction that all but only leaves initially have an assigned processor. The numbers along the edges express the according costs.

Then, we compute recursively for each node $v \in V \setminus le(V)$:

$$pen_i(v) = \sum_{u \in pa(v)} \min_{j \in P} \{f_u \cdot d_{j,i} + pen_j(u)\}.$$

Figure 3.a) shows the penalty values obtained from applying the inward phase on the instance given in figure 2.

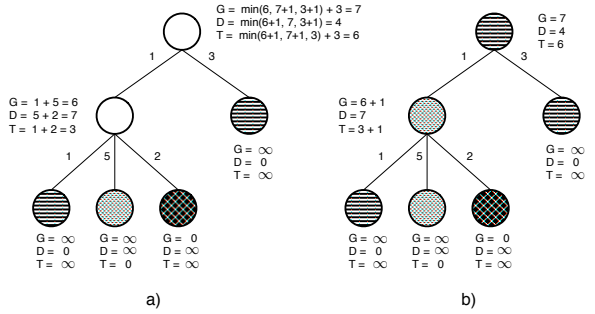

Figure 3. a) The penalty values computed during the inward phase. The capitals stand for: G = *grilled*, D = *dashed* and T = *dotted*. b) The completed processor assignment that minimizes communication costs.

Phase II: We now determine a complete processor assignment mapping $\xi : V \to P$ such that the total communication costs in the join tree are minimized. For the root node $r \in V$, we define $\xi(r) = i$ such that $pen_i(r) = \min$. Then we assign recursively to each other node $v \in V$, $\xi(v) = i$ if $f_v \cdot d_{i,\xi(ch(v))} + pen_i(v) \leq f_v \cdot d_{j,\xi(ch(v))} + pen_j(v)$ for all $i, j \in P$.

Note that whenever a new node is considered during the outward phase, it is ensured that a processor has already been assigned to its descendants. Therefore, the recursive minimization step of phase two is well-defined. Figure 3.b) finally shows how processors are assigned to join tree nodes regarding the penalty values computed during the inward phase.

It is easy to see that at the end of the algorithm we have $\xi(v) = \chi(\psi_v)$ for all $v \in le(V)$. Since G is a tree, there are $|V| - 1$ edges and therefore the algorithm consists of roughly $2 \cdot |V|$ steps. At each step, we compute $|P|^2$ sums and take the minimum, which results in a time complexity of $O(|V| \cdot |P|^2)$.

5 CONCLUSION

Minimization of communication costs is a central topic in the analysis of distributed algorithm and it is often the case that this process induces optimization problems with intractable complexity. At all times, local computation architectures are described as distributed algorithms where join tree nodes act as virtual processors and exchange messages to contribute to a common computational task. But with the focus on the distributed nature of knowledge, these virtual processors become real in a very natural way. This paper shows that in such a distributed computing environment, the costs of transmitting local computation messages can be minimized with small polynomial effort. Thereby, weight predictability turned out to be a sufficient condition for the underlying valuation algebra. These insights will be a central point in the further development of local computation software, headed up by NENOK [9], which is based on a comparable computing environment and provides already corresponding implementation of this theory.

ACKNOWLEDGEMENT

I am grateful to Prof. Dr. Jürg Kohlas for his support and to the anonymous referee for his valuable comments.

REFERENCES

[1] E. Dahlhaus, D. S. Johnson, C. H. Papadimitriou, P. D. Seymour, and M. Yannakakis, 'The complexity of multiterminal cuts', *SIAM J. Comput.*, **23**(4), 864–894, (1994).

[2] Peter L. Erdös and Laszlo A. Szekely, 'On weighted multiway cuts in trees', *Math. Program.*, **65**(1), 93–105, (1994).

[3] J. Kohlas, 'Information in context', Technical Report 02-15, Department of Informatics, University of Fribourg, (2002).

[4] J. Kohlas, *Information Algebras: Generic Structures for Inference*, Springer-Verlag, 2003.

[5] J. Kohlas, R. Haenni, and S. Moral, 'Propositional information systems', *Journal of Logic and Computation*, **9 (5)**, 651–681, (1999).

[6] J. Kohlas and N. Wilson, 'Exact and approximate local computation in semiring induced valuation algebras', Technical Report 06-06, Department of Informatics, University of Fribourg, (2006).

[7] N. Lehmann, *Argumentation System and Belief Functions*, Ph.D. dissertation, Department of Informatics, University of Fribourg, 2001.

[8] J. Mengin and N. Wilson, 'Logical deduction using the local computation framework', in *European Conf. ECSQARU'99, London*, eds., A. Hunter and S. Parsons, Lecture Notes in Artif. Intell., pp. 386–396. Springer, (1999).

[9] M. Pouly, 'Nenok 1.1 user guide', Technical Report 06-02, Department of Informatics, University of Fribourg, (2006).

[10] M. Pouly and J. Kohlas, 'Minimizing communication costs of distributed local computation', Technical Report 05-20, Department of Informatics, University of Fribourg, (2005).

[11] C. Schneuwly, M. Pouly, and J. Kohlas, 'Local computation in covering join trees', Technical Report 04-16, Department of Informatics, University of Fribourg, (2004).

[12] G. Shafer, 'An axiomatic study of computation in hypertrees', Working Paper 232, School of Business, University of Kansas, (1991).

[13] P. P. Shenoy and G. Shafer, 'Axioms for probability and belief-function proagation', in *Uncertainty in Artificial Intelligence 4*, eds., Ross D. Shachter, Tod S. Levitt, Laveen N. Kanal, and John F. Lemmer, volume 9 of *Machine intelligence and pattern recognition*, pp. 169–198, Amsterdam, (1990). Elsevier.