Towards Automatic Measurement of Probabilistic Processes[*]

Lin Song, Yuxin Deng, Xiaojuan Cai
Department of Computer Science, Shanghai Jiao Tong University, China
800 Dongchuan Road, Shanghai 200240, China
{colin_05, yuxindeng, cxj}@sjtu.edu.cn

Abstract

In this paper we propose a metric for finite processes in a probabilistic extension of CSP. The kernel of the metric corresponds to trace equivalence and most of the operators in the process algebra is shown to satisfy non-expansiveness property with respect to this metric. We also provide an algorithm to calculate the distance between two processes to a prescribed discount factor in polynomial time. The algorithm has been implemented in a tool that helps us to measure processes automatically.

1 Introduction

Quantitative transition systems extend the usual transition systems, by labeling transitions with both actions and numbers. It has been argued that notions of exact behavioral equivalence such as trace equivalence do not fit well with quantitative transition systems. The standard notion of trace equivalence treat the quantities as labels, this means that processes that differ from a very small probability would be considered just as different as processes that perform completely different actions. To find a more flexible way to identify processes, we want an "approximate" notion of equality of processes. Researchers have used the notion of metric to express the similarity of the behavior of processes. The smaller the distance, the more alike the behavior is. In particular, the distance between two processes is 0 if they are indistinguishable.

Much work has been reported recently in this field. The first proposal based on metric was made by Giacalone et al. [10] for deterministic probabilistic processes. Later, van Breugel et al. [19, 18] and Desharnais et al. [9, 8] investigated the notion of metric for more general probabilistic systems involving probabilistic distributions, nondeterminism and recursion. In [19, 18], van Breugel and Worrell defined a metric via the terminal coalgebra of a functor and studied the fine structure of their metric whose dual problem can be reduced to a particular linear programming problem: the transshipment problem. In [9, 8], Desharnais et al. studied the metrics for labeled Markov chains and labeled concurrent Markov chains respectively and defined the intended metric as the greatest fixed point of a monotonous function. Deng et al. [4] defined a notion of state-metric, which extended the approach of [9, 8], to a more general framework called action-labeled quantitative transition systems.

In this paper, we measure the distance of processes in a finite probabilistic process algebra, pCSP, investigated in [7, 6]. We represent the operational behavior of a process as a finite acyclic directed graph and interpret it as a set of distributions on maximal traces. We first define a notion of metric for traces. Then we lift it to distributions of traces by Kantorovich metric [12]. Next, we lift it again to be a metric for sets of distributions of traces using Hausdorff distance. Finally, the distance of two processes is determined by the distance of two sets of distributions of traces that they represent. We show that the kernel of the metric corresponds to trace equivalence and all operators of the process algebra, except for parallel composition, satisfy *non-expansiveness* property with respect to this metric, i.e. the distance between two processes does not expand when they are put in the same context. We also give an algorithm to calculate the distance between two processes to a prescribed $discount\ factor$ in polynomial time. We have developed an tool that implements this algorithm and helps to measure processes automatically.

The rest of the paper is organized as follows. In Section 2, we introduce pCSP and give the notion of trace equivalence between processes. In Section 3, we describe our metric on processes and relate it to trace equivalence. Some non-expansiveness properties are proved in Section 4. The reduction of calculating the metric to a transshipment problem is presented in Section 5. In Section 6, we present an algorithm. Some examples and comparison with other metrics are discussed in Section 7. In Section 8, we conclude and discuss some future work.

[*]Supported by The National 973 Project (2003CB317005); The National Nature Science Foundation of China (60473006,60573002); The BoShiDian Research Fund (20010248033).

2 Finite Probabilistic CSP

We briefly introduce a simple probabilistic process algebra based on CSP [1]; following [7, 6], we call it pCSP. We give its operational semantics in terms of a probabilistic labeled transition system and define a notion of trace equivalence between processes.

2.1 Syntax and Semantics

Let Act be a finite set of actions, ranged over by $a, b, \cdots$, which processes can perform. Then the finite probabilistic CSP processes are given by the following two-sorted syntax:

$$\begin{array}{lcl} P & ::= & S \mid P\,{}_p\!\oplus P \\ S & ::= & \mathbf{0} \mid a.P \mid P \sqcap P \mid S \,\Box\, S \mid S \mid_A S \end{array}$$

Here $P \;{}_p\!\oplus\; Q$, for $p \in (0, 1]$, represents a *probabilistic choice* between P and Q: with probability p it will act like P and with probability $1 - p$ it will act like Q. Any process is a probabilistic combination of state-based processes (the sub-sort S above) built by repeated application of the operator ${}_p\!\oplus$. The state-based processes have a CSP-like syntax, involving the stopped process $\mathbf{0}$, action prefixing $a.\_$, *internal-* and *external choices* $\sqcap$ and $\Box$, and a *parallel composition* $|_A$ for $A \subseteq Act$.

The process $P \sqcap Q$ will first do a so-called *internal action* $\tau \notin Act$, choosing *nondeterministically* between P and Q. Therefore $\sqcap$, like $a.\_$, acts as a *guard*, in the sense that it converts any process arguments into a state-based process.

The process $P \,\Box\, Q$ on the other hand does not perform actions itself, but merely allows its arguments to proceed, disabling one argument as soon as the other has done a visible action. In order for this process to start from a state rather than a probability distribution of states, we require its arguments to be state-based as well; the same applies to $|_A$. Expressions $P \,\Box\, Q$ and $P \;|_A\; Q$ for processes P and Q that are *not* state-based are therefore syntactic sugar for an expression in the above syntax obtained by distributing $\Box$ and $|_A$ over ${}_p\!\oplus$.

Finally, the expression $P \;|_A\; Q$, where $A \subseteq Act$, represents processes P and Q running in parallel. They may synchronize by performing the same action from A simultaneously; such a synchronization results in τ. In addition P and Q may independently do any action from $(Act\backslash A) \cup \{\tau\}$.

We write pCSP for the set of process terms defined by this grammar, and sCSP for the subset comprising only the state-based process terms. The full language of CSP [1] has many more operators; we have simply chosen a representative selection, and have added probabilistic choice. Our parallel operator is not a CSP primitive, but it can easily be expressed in terms of them. It can also be expressed in terms of the parallel composition, renaming and restriction operators of CCS.

As usual the prefixing operator $a.\_$ binds stronger than any binary operator; and precedence between binary operators is indicated via brackets or spacing.

The above intuitions are formalised by an *operational semantics* associating with each process term a graph-like structure representing its possible reactions to users' requests: we use a generalisation of labelled transition systems [1] that includes probabilities.

A (discrete) probability distribution over a set S is a function $\Delta : S \rightarrow [0, 1]$ with $\sum_{s \in S} \Delta(s) = 1$; the *support* of Δ is given by $\lceil \Delta \rceil = \{\, s \in S \mid \Delta(s) > 0 \,\}$. We write $\mathcal{D}(S)$, ranged over by Δ, for the set of all distributions over S with finite support. We also write $\overline{s}$ to denote the point distribution assigning probability 1 to s and 0 to all others, so that $\lceil \overline{s} \rceil = \{s\}$.

We now give the probabilistic generalisation (pLTSs) of labelled transition systems (LTSs):

Definition 1. *A* probabilistic labelled transition system *is a triple* $\langle S, Act_\tau, \rightarrow \rangle$, *where*

1. *S is a set of states;*
2. *Act_τ is a set of actions Act, augmented by $\tau \notin Act$; we let a range over Act and α over Act_τ;*
3. *relation $\rightarrow$ is a subset of $S \times Act_\tau \times \mathcal{D}(S)$.*

As with LTSs, we usually write $s \xrightarrow{\alpha} \Delta$ for $(s, \alpha, \Delta) \in \rightarrow$. An LTS may be viewed as a degenerate pLTS, one in which only point distributions are used.

We now define the operational semantics of pCSP by means of a particular pLTS $\langle \mathsf{sCSP}, Act_\tau, \rightarrow \rangle$, constructed by taking sCSP to be the set of states and interpreting pCSP processes P as distributions $[\![P]\!] \in \mathcal{D}(\mathsf{sCSP})$ as follows:

$$\begin{array}{rcl} [\![s]\!] & := & \overline{s} \quad\quad \text{for } s \in \mathsf{sCSP} \\ [\![P\,{}_p\!\oplus Q]\!] & := & [\![P]\!]\,{}_p\!\oplus [\![Q]\!]. \end{array}$$

Here we write $\Delta_1 \,{}_p\!\oplus\, \Delta_2$ for the linear combination of two distributions: $p \cdot \Delta_1 + (1 - p) \cdot \Delta_2$. Note that for each $P \in \mathsf{pCSP}$ the distribution $[\![P]\!]$ is finite, i.e. it has finite support. The definition of the relations $\xrightarrow{\alpha}$ is given in Figure 1. These rules are very similar to the standard ones used to interpret CSP as an LTS [1], but modified so that the result of an action is a distribution. The symmetric rules have all been omitted.

2.2 PAD and NPAD graphs

We graphically depict the operational semantics of a pCSP expression P by drawing the part of the pLTS defined above that is reachable from $[\![P]\!]$ as a finite graph,

$$a.P \xrightarrow{a} [\![P]\!]$$

$$P \sqcap Q \xrightarrow{\tau} [\![P]\!]$$

$$\frac{s_1 \xrightarrow{\tau} \Delta}{s_1 \Box s_2 \xrightarrow{\tau} \Delta \Box \overline{s_2}}$$

$$\frac{s_1 \xrightarrow{a} \Delta}{s_1 \Box s_2 \xrightarrow{a} \Delta}$$

$$\frac{s_1 \xrightarrow{\alpha} \Delta \quad \alpha \notin A}{s_1|_A s_2 \xrightarrow{\alpha} \Delta|_A \overline{s_2}}$$

$$\frac{s_1 \xrightarrow{a} \Delta_1, s_2 \xrightarrow{a} \Delta_2 \quad a \in A}{s_1|_A s_2 \xrightarrow{\tau} \Delta_1|_A \Delta_2}$$

Figure 1. Operational semantics of pCSP.

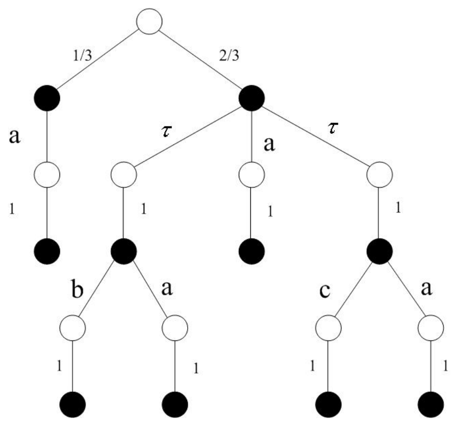
which we call nondeterministic probabilistic acyclic directed (NPAD) graph. Our NPAD graphs involve two sorts of nodes, states and distributions, denoted by • and ∘ respectively. For any state s and distribution Δ with $s \xrightarrow{\alpha} \Delta$ we draw an edge from s to Δ, labeled with α. For any distribution Δ and state s in $\lceil \Delta \rceil$, the support of Δ, we draw an edge from Δ to s, labeled with $\Delta(s)$. Therefore, NPAD graphs are depicted with edges labeled by actions and probabilities alternately. In general, a state may have multiple outgoing distributions because of nondeterminism.

We now consider execution trees of an NPAD graph under particular *adversaries* which help resolving nondeterminism. The approach we adopt here is similar to the one proposed in [17] for probabilistic automata. The idea behind is that at most one action is allowed for each state. More precisely, an adversary for an NPAD graph is a function that associates to each state a distribution which is selected. An execution tree, which we call probabilistic acyclic directed (PAD) graph, is obtained from an NPAD graph by pruning all the edges corresponding to distributions which are not selected by the adversary.

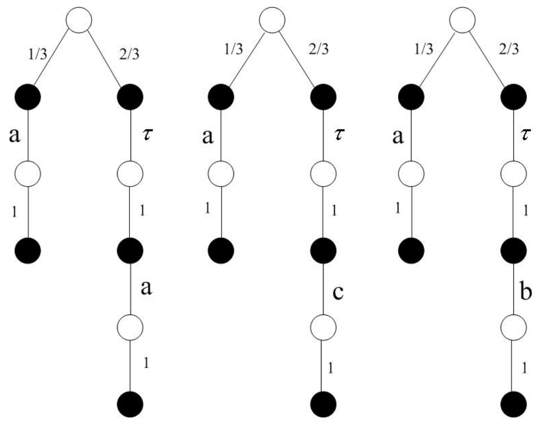
Given a pCSP expression P, we use $(\![P]\!)$ to denote the set of PAD graphs obtained from the NPAD graph which represents P. Consider the process

$$P = a.0 \,_{1/3}\!\oplus\, ((b.0 \sqcap c.0) \Box a.0).$$

Its NPAD graph and some PAD graphs are illustrated in Figure 2 (a) and (b) respectively. Details on how to obtain a set of PAD graphs from an NPAD graph are discussed in Section 6.



(a) An NPAD graph



(b) Some PAD graphs

Figure 2. NPAD/PAD graphs

2.3 Trace equivalence

Given a PAD graph $\mathcal{G}$, we can consider it as a set of strings $ss(\mathcal{G})$ obtained from each maximal path, that is, a set of strings of the form of $p_0, \alpha_1, p_1, \alpha_2 ..., \alpha_n, p_n$ where $\alpha_i \in Act_\tau$ and $p_i \in (0, 1]$. We introduce two mappings on $ss(\mathcal{G})$.

Definition 2. *Let* $\Phi : ss(\mathcal{G}) \to Act_\tau^*$ *and* $\Psi : ss(\mathcal{G}) \to [0, 1]$ *be the mapings defined as follows. For any* $x = p_0, \alpha_1, p_1, \alpha_2 ..., \alpha_n, p_n \in ss(\mathcal{G})$*:*

$$\Phi(x) = \begin{cases} \langle\rangle & if\ n = 0 \\ \alpha_1\alpha_2...\alpha_n & otherwise. \end{cases}$$

$$\Psi(x) = \begin{cases} p_0 & if\ n = 0 \\ p_0 \cdot p_1 \cdots p_n & otherwise. \end{cases}$$

where $\langle\rangle$ denotes the empty path.

For any PAD graph $\mathcal{G}$ and $x \in ss(\mathcal{G})$, the above definition says that $\Phi(x)$ is a path in $\mathcal{G}$ and $\Psi(x)$ is the probability of performing this path. A path is maximal if there is no outgoing distributions from the last state of the path. In this paper, we insist that all the paths appeared are maximal. The trace of a path is the sequence obtained by restricting the actions of a path to the set Act. We use $\Phi'(x)$ to denote the trace of the path and $ts(\mathcal{G})$ stands for the set of traces $\mathcal{G}$ can performs.

Let $\mathcal{G}$ be the first PAD graph given in Figure 2 (b). We can represent $\mathcal{G}$ as the following set of strings: $\{1/3, a, 1, \;\; 2/3, \tau, 1, a, 1\}$. By the definition of Φ' we have $\Phi'(1/3, a, 1) = \Phi'(2/3, \tau, 1, a, 1) = a$. Thus the probability with which the PAD graph performs the trace a will be $\Psi(1/3, a, 1) + \Psi(2/3, \tau, 1, a, 1) = 1$. More formally, we have the following definition.

Definition 3. *For any PAD graph $\mathcal{G}$ and trace u, let $\nu(\mathcal{G}, u)$ denote the probability of $\mathcal{G}$ performing the trace u, defined as follows:*

$$\nu(\mathcal{G}, u) = \begin{cases} 0 & \text{if } \Phi'(x) \neq u \\ \Sigma_{\Phi'(x)=u}\Psi(x) & otherwise. \end{cases}$$

where $x \in ss(\mathcal{G})$

We introduce the notion of *trace distribution*, originally proposed by Segala for probabilistic automata [16]. A trace distribution over a set $ts(\mathcal{G})$ is a mapping $\Theta : ts(\mathcal{G}) \to [0, 1]$ such that $\Sigma_{u \in ts(\mathcal{G})}\nu(\mathcal{G}, u) = 1$. A trace distribution corresponds to a discrete probability distribution on the maximal traces of a PAD graph. Two PAD graphs $\mathcal{G}, \mathcal{G}'$ are trace equivalent if they have the same trace distribution, denoted $\mathcal{G} \approx_{tr} \mathcal{G}'$. We lift this definition to two sets of PAD graphs Ω, Ω' by saying that $\Omega \approx_{tr} \Omega'$ if for each PAD graph in Ω there exists a trace equivalent PAD graph in Ω', and vice versa. Since we interpret an NPAD graph as a set of PAD graphs, it is then natural to say that two NPAD graphs are trace equivalent if they generate two sets of trace equivalent PAD graphs.

Definition 4. *Two* pCSP *processes P and Q are said to be* trace equivalent, *written $P \approx_{tr} Q$, if $[\![P]\!] \approx_{tr} [\![Q]\!]$.*

3 Metrics

We define in this section a metric for pCSP processes, and we relate it to trace equivalence such that two processes have distance zero under the metric if and only if they are trace equivalent.

We first introduce the general notation of (pseudo)metric. A metric over a set S is a function that yields a non-negative real number between 0 and 1 for each pair of elements from S and satisfies the following: $m(s, s) = 0$, $m(s, t) = m(t, s)$ and $m(s, t) \leq m(s, u) + m(u, t)$.

3.1 A Metric for Traces

First we define a metric for traces. This metric balances the depth of observations which is needed to distinguish the traces and the degree to which each observation differentiates the traces. The relative weight given to these two factors is determined by a discount factor c. The smaller the value of c the greater the discount on observations made at greater depth. From this intuition, we can now define our metric for traces, which is a variation on the Baire metric.

Definition 5. *Let u, u' be traces and c be a real number lying between 0 and 1.*

$$m_1(u, u') = \begin{cases} c^{k-1} & \text{if } u[k] \neq u'[k] \text{ and } u[i] = u'[i] \\ & \quad \text{for } (i < k) \\ 0 & otherwise. \end{cases}$$

where $u[i]$ means the i-th action of the trace u.

The metric for traces discounts the distance at position i of the trace by multiplying it by c^{i-1}. We consider the following traces $u = abce$ and $u' = abcd$ which differ on their fourth actions. Calculating the distance we have $m_1(u, u') = 1/8$ if c is supposed to be $1/2$. Now we prove that the above metric is well defined.

Proposition 1. *m_1 is a metric for traces.*

Proof. We need to verify that m_1 satisfies the three properties stated in the introduction of this section.

1. For all u, u', $m_1(u, u') \geq 0$ and $m_1(u, u) = 0$ follows from the definition of m_1.

2. For all u, u', $m_1(u, u') = m_1(u', u)$ holds trivially.

3. $m_1(u, u'') + m_1(u'', u') \geq m_1(u, u')$ is proved as follow: There exists five cases including $u = u' = u''$, $u = u' \neq u''$, $u \neq u' = u''$, $u'' = u \neq u'$ and $u \neq u' \neq u''$. The first four ones are straightforward. Here we only prove the last one. Suppose $m_1(u, u'') = c^{k_1-1}$ and $m_1(u'', u') = c^{k_2-1}$, without loss of generality, $k_1 \leq k_2$. Then $m_1(u, u') = c^{k-1}$ such that $k = k_1$. It follows that $m_1(u, u'') + m_1(u'', u') \geq m_1(u, u')$.

□

3.2 A Metric for PAD Graphs

We now define a metric for PAD graphs. Essentially, it is obtained by lifting the above metric for traces m_1 to be a metric for trace distributions m_2 based on Kantorovich metric [12].

Definition 6. *Let $\mathcal{G}$, $\mathcal{G}'$ be PAD graphs, δ represent $ts(\mathcal{G}) \cup ts(\mathcal{G}')$. Then $m_2(\mathcal{G}, \mathcal{G}')$ is defined as follows:*

1. *if $\Sigma_{u_i \in \delta}(\nu(\mathcal{G}, u_i) - \nu(\mathcal{G}', u_i)) \geq 0$, then $m_2(\mathcal{G}, \mathcal{G}')$ is given by the solution to the following linear program:*

$$\begin{array}{ll} \textit{maximize} & \Sigma_{u_i \in \delta}(\nu(\mathcal{G}, u_i) - \nu(\mathcal{G}', u_i))x_i \\ \textit{subject to} & \forall i : x_i \geq 0 \\ & \forall i, j : x_i - x_j \leq m_1(u_i, u_j) \end{array}$$

2. *otherwise, $m_2(\mathcal{G}, \mathcal{G}')$ is defined as $m_2(\mathcal{G}', \mathcal{G})$.*

Let us consider the second and third PAD graphs given in Figure 2 (b). By the definition of m_2 we have that

$$\begin{array}{ll} \textit{maximize} & 0\,x_1 + 2/3\,x_2 - 2/3\,x_3 \\ \textit{subject to} & \forall i : x_i \geq 0 \\ & \forall i, j : x_i - x_j \leq m_1(u_i, u_j) = 1 \end{array}$$

Using the above definition we have the following proposition connecting trace equivalence and the metric for PAD graphs.

Proposition 2. *For any PAD graphs $\mathcal{G}$ and $\mathcal{G}'$,*

$$\mathcal{G} \approx_{tr} \mathcal{G}' \Leftrightarrow m_2(\mathcal{G}, \mathcal{G}') = 0.$$

Proof. ($\Rightarrow$) Suppose $\mathcal{G} \approx_{tr} \mathcal{G}'$. By the definition of trace equivalence we can conclude that $\nu(\mathcal{G}, u) = \nu(\mathcal{G}', u)$ for any trace u. It follows that every coefficient of the linear program in Definition 6 is 0 and thus $m_2(\mathcal{G}, \mathcal{G}') = 0$ is proved.

($\Leftarrow$) Suppose $m_2(\mathcal{G}, \mathcal{G}') = 0$. To show $\mathcal{G} \approx_{tr} \mathcal{G}'$ we need to prove $\nu(\mathcal{G}, u) = \nu(\mathcal{G}', u)$ for any trace u. In order to ensure $m_2(\mathcal{G}, \mathcal{G}') = 0$, the following two conditions must be satisfied. Before proceeding with the proof we use the abbreviation that $a_i = \nu(\mathcal{G}, u_i) - \nu(\mathcal{G}', u_i)$ for each trace u_i. Then the linear function of m_2 can be reformulated to be $\Sigma_{u_i \in \delta}\, a_i \cdot x_i$.

1. No coefficient is positive. Otherwise, if $a_i > 0$ for a certain trace u_i, the solution of the linear program is a value $a_i \cdot x_i$, which is larger than 0 as long as $x_i > 0$.

2. It is not case that at least one coefficient is negative and the other coefficients are either negative or 0. Otherwise, by summing all the coefficients, we would get $\Sigma_{u_i \in \delta}\, a_i < 0$ which contradicts the definition of m_2.

Therefore, every coefficient of the linear program is 0, which leads to $\nu(\mathcal{G}, u) = \nu(\mathcal{G}', u)$ for any trace u. Therefore, we have $\mathcal{G} \approx_{tr} \mathcal{G}'$. □

3.3 A Metric for Probabilistic Processes

We lift the metric for PAD graphs m_2 to be a metric for sets of PAD graphs m_3, by means of the Hausdorff distance discussed in [14].

Definition 7. *Let Ω, Ω' be two sets of PAD graphs.*

$$m_3(\Omega, \Omega') = \\ \max(\sup_{\mathcal{G} \in \Omega} \inf_{\mathcal{G}' \in \Omega'} m_2(\mathcal{G}, \mathcal{G}'), \sup_{\mathcal{G}' \in \Omega'} \inf_{\mathcal{G} \in \Omega} m_2(\mathcal{G}', \mathcal{G}))$$

We are now in a position to define a metric for pCSP processes.

Definition 8. *Let P, Q be two* pCSP *processes,*

$$m(P, Q) = m_3([\![P]\!], [\![Q]\!]).$$

We give a counterpart of Proposition 2 for processes.

Proposition 3. *For any two P, Q,*

$$P \approx_{tr} Q \Leftrightarrow m(P, Q) = 0.$$

Proof. ($\Rightarrow$) Suppose $P \approx_{tr} Q$. By the definition of trace equivalence, we see that $[\![P]\!]$ and $[\![Q]\!]$ have the same collection of trace distributions. It is easy to indicate that for any $\mathcal{G} \in [\![P]\!]$ there exists a $\mathcal{G}' \in [\![Q]\!]$ which can meet $\mathcal{G} \approx_{tr} \mathcal{G}'$. This means that $m_2(\mathcal{G}, \mathcal{G}') = 0$, and vice versa. It follows that $m_3([\![P]\!], [\![Q]\!]) = 0$ and therefore $m(P, Q) = 0$.

($\Leftarrow$) Suppose $m(P, Q) = 0$. This means that for any $\mathcal{G} \in [\![P]\!]$ we can find a $\mathcal{G}' \in [\![Q]\!]$ such that $m_2(\mathcal{G}, \mathcal{G}') = 0$, and vice versa. So P and Q have the same collection of trace distributions modulo trace equivalence. □

4 Non-expansiveness

We show in this section that prefixing and the three kinds of choice operators of pCSP are non-expansive, which means that different processes are more similar when they are put in the same context. However, we require that the context is constructed from all other operators except for parallel composition. This is not surprising because trace equivalence is in general not a congruence relation with respect to parallel composition [11].

Proposition 4. *For any process P, Q, R, if $m(P, Q) \leq \varepsilon$ then*

1. $m(a.P, a.Q) \leq \varepsilon$
2. $m(P \sqcap R, Q \sqcap R) \leq \varepsilon$
3. $m(P \Box R, Q \Box R) \leq \varepsilon$
4. $m(P \,{}_p{\oplus}\, R, Q \,{}_p{\oplus}\, R) \leq \varepsilon$.

Proof. Given a process P, we write $\mathcal{P}(P)$ for any PAD graph in $[\![P]\!]$.

- As we know, $m_2(\mathcal{P}(a.P), \mathcal{P}(a.Q))$ is the solution to the follow linear program:

$$\begin{array}{ll} \multicolumn{2}{l}{maximize} \\ \multicolumn{2}{l}{\Sigma_{a.u_i \in \delta}(\nu(\mathcal{P}(a.P), a.u_i) - \nu(\mathcal{P}(a.Q), a.u_i))x_i} \\ subject\ to & \forall i : x_i \geq 0 \\ & \forall i, j : x_i - x_j \leq m_1(a.u_i, a.u_j) \end{array}$$

  where

$$\nu(\mathcal{P}(a.P), a.u_i) - \nu(\mathcal{P}(a.Q), a.u_i) \\ = \nu(\mathcal{G}, u_i) - \nu(\mathcal{G}', u_i)$$

  and $m_1(a.u_i, a.u_j) \leq m_1(u_i, u_j)$. It follows that $m_2(\mathcal{P}(a.P), \mathcal{P}(a.Q)) \leq m_2(\mathcal{P}(P), \mathcal{P}(Q))$. By the definition of m we have that

$$m(a.P, a.Q) \leq m(P, Q) \leq \varepsilon.$$

- Let $P + R$ denote $P \sqcap Q$ and $P \Box Q$. Before giving a proof of the properties we use the following notion that $m_3'([\![P]\!], [\![Q]\!]) = \sup_{\mathcal{G} \in [\![P]\!]} \inf_{\mathcal{G}' \in [\![Q]\!]} m_2(\mathcal{G}, \mathcal{G}')$.

$$\begin{aligned} & m_3'([\![P + R]\!], [\![Q + R]\!]) \\ = & \sup_{\mathcal{G} \in [\![P]\!] \cup [\![R]\!]} \inf_{\mathcal{G}' \in [\![Q]\!] \cup [\![R]\!]} m_2(\mathcal{G}, \mathcal{G}') \\ = & \sup_{\mathcal{G} \in [\![P]\!]} \inf_{\mathcal{G}' \in [\![Q]\!] \cup [\![R]\!]} m_2(\mathcal{G}, \mathcal{G}') \\ \leq & \sup_{\mathcal{G} \in [\![P]\!]} \inf_{\mathcal{G}' \in [\![Q]\!]} m_2(\mathcal{G}, \mathcal{G}') \\ = & m_3'([\![P]\!], [\![Q]\!]) \end{aligned}$$

  Similarly,

$$m_3'([\![Q + R]\!], [\![P + R]\!]) \quad \leq \quad m_3'([\![Q]\!], [\![P]\!])$$

  Therefore, we can infer that

$$\begin{aligned} & m(P + R, Q + R) \\ & \leq \max(m_3'([\![P]\!], [\![Q]\!]), m_3'([\![Q]\!], [\![P]\!])) \\ & = m(P, Q) \\ & \leq \varepsilon \end{aligned}$$

- $m_2(\mathcal{P}(P\ {}_p\oplus\ R), \mathcal{P}(Q\ {}_p\oplus\ R))$ is the solution to the following linear program:

$$\begin{array}{ll} \multicolumn{2}{l}{maximize} \\ \multicolumn{2}{l}{\Sigma_{u_i \in \delta}(\nu(\mathcal{P}(P\ {}_p\oplus\ R), u_i) - \nu(\mathcal{P}(Q\ {}_p\oplus\ R), u_i))x_i} \\ subject\ to & \forall i : x_i \geq 0 \\ & \forall i, j : x_i - x_j \leq m_1(u_i, u_j) \end{array}$$

  where

$$\Sigma_{u_i \in \delta}(\nu(\mathcal{P}(P\ {}_p\oplus\ R), u_i) - \nu(\mathcal{P}(Q\ {}_p\oplus\ R), u_i))x_i \\ = p(\Sigma_{u_i \in \delta'}(\nu(\mathcal{G}, u_i) - \nu(\mathcal{G}', u_i))x_i)$$

  and $\delta' = \delta - ts(\mathcal{P}(R))$. The dual of m_2 can be reduced to a transshipment problem (cf. Section 5).

  We can find that these two transshipment problems reduced from

$$m_2(\mathcal{P}(P\ {}_p\oplus\ R), \mathcal{P}(Q\ {}_p\oplus\ R))$$

  and $m_2(\mathcal{P}(P), \mathcal{P}(Q))$ have the same sinks, sources as well as costs between these nodes. One difference between the two transshipment problems is that the first problem has some intermediate nodes while the second one has no such nodes. This means that it may be more expensive to send units directly from a given source to a given sink, rather than indirectly. Another difference lies in the fact that the demand of the first problem is that of the second one with a discount factor p. Then we can conclude

$$m_2(\mathcal{P}(P\ {}_p\oplus\ R), \mathcal{P}(Q\ {}_p\oplus\ R)) \\ \leq p * m_2(\mathcal{P}(P), \mathcal{P}(Q))$$

  and thus

$$\begin{aligned} & m(P\ {}_p\oplus\ R, Q\ {}_p\oplus\ R) \\ & \leq p * m(P, Q) \\ & = p * \varepsilon \\ & \leq \varepsilon \end{aligned}$$

$\Box$

However, parallel composition is not non-expansive. Consider the following example: $P = a.(b.0\ {}_{1/2}\oplus\ c.0)$, $Q = a.b.0\ {}_{1/3}\oplus\ a.c.0$ and $R = a.0$. It can be checked that $m(P|_{\{a\}}R, Q|_{\{a\}}R) = 1/6$ while $m(P, Q) = 1/12$.

5 Linear Programming

In this section, we show that the metric m_2 between trace distributions can be reduced to a transshipment problem. For more discussions on this problem we refer the reader to the textbook of Chvatal [2]. The transshipment problem is to find the cheapest way to ship a prescribed amount of a commodity from specified origins to specified destinations through a concrete transportation network. This network is represented by a directed graph.

The metric in Definition 6 is given in terms of a linear programming problem and if all x_i are 0 this problem has a feasible origin. According to the fundamental theorem of linear programming, this problem has an optimal solution as the dual problem. If $N + 1$ denotes the number of traces in δ, then the dual problem of m_2 can be formulated in terms of matrices and vectors as follows.

$$minimize \quad cost \cdot vector^T$$

such that

$$matrix \cdot vector^T = demand^T$$

and

$$vector \geq 0$$

where the $(N^2 + N)$-vector $cost$ is defined by

$$cost_l = \begin{cases} m_1(u_{(l\ mod\ N)}, u_{(l\ div\ N)}) & \text{if } condition \\ m_1(u_N, u_{(l\ div\ N)}) & \text{othersize.} \end{cases}$$

in which $condition$ stands for the condition

$$0 \leq l < N^2 + N \text{ and } l\ mod\ N \neq l\ div\ N$$

and the $(N + 1)$-vector $demand$ is defined by

$$demand_k = \nu(\mathcal{G}, u_k) - \nu(\mathcal{G}', u_k) \quad 0 \leq k \leq N$$

and the $(N + 1) \times (N^2 + N)$-matrix $matrix$ is defined by

$$matrix_{k,l} = \begin{cases} 1 & \text{if } (k = l\ mod\ N \wedge k \neq l\ div\ N) \\ & \text{or } (k = N \wedge l\ mod\ (N+1) = 0) \\ -1 & \text{if } k = l\ div\ N \\ 0 & \text{othersize.} \end{cases}$$

Consequently, the definition of m_2 is an instance of the transshipment problem.

For the second and third PAD graphs given in Figure 2 (b), calculating their distance amounts to solving the following transshipment problem.

$$\begin{array}{ll} minimize & (1,1,1,1,1,1) \cdot vector^T \\ subject\ to & matrix \cdot vector^T = demand^T \\ & vector \geq 0 \end{array}$$

where matrix and demand looks as follows.

$$\begin{array}{rcl} matrix & = & \begin{pmatrix} -1 & -1 & 1 & 0 & 1 & 0 \\ 0 & 1 & -1 & -1 & 0 & 1 \\ 1 & 0 & 0 & 1 & -1 & -1 \end{pmatrix} \\ demand & = & (0, 2/3, -2/3) \end{array}$$

6 Algorithm

Given two pCSP processes P and Q, we present an algorithm to calculate the distance $m(P, Q)$. The presentation is aimed at clarity. For more details we refer the reader to Appendix A.

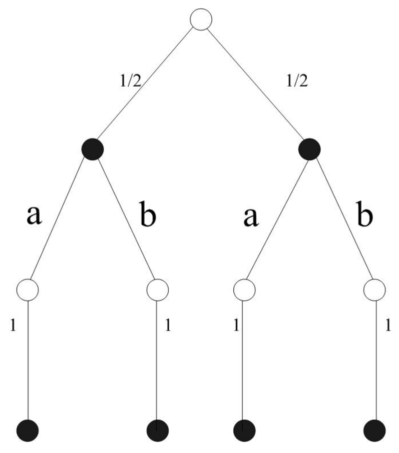
Our algorithm implements m_2 by the procedure $Distance$ which returns a real number between [0,1]. Efficient implementation of $Distance$ is crucial to the overall complexity of the algorithm. For two PAD graphs, we obtain a set $\mathscr{A}$ of pairs of traces and probabilities of the form (u, p) by computing $\nu(\mathcal{G}, u) - \nu(\mathcal{G}', u)$. $N+1$ denotes the number of elements in the set $\mathscr{A}$. $\mathbb{p}(u)$ denotes the second part of the pair (u, p) for any trace u. We have proved that the dual problem of m_2 can be reduced to a transshipment problem. Then we can construct the dual problem of m_2 and initiate each parameter of this transshipment problem step by step according to the proof in Section 5. Finally, we can solve the transshipment problem using the polynomial dual network simplex algorithm introduced in [15]. It should be noted that we require the discount factor c when computing parameter $cost$.

In order to get PAD graphs, we need to have a procedure *split* which splits a NPAD graph into a set of PAD graphs. Given a process P, we write $\mathcal{NP}(P)$ for any NPAD graph. In this procedure, we use sets $\mathscr{M}$ and $\mathscr{N}$ to contain NPAD graphs and PAD graphs, respectively. First, this procedure depicts NPAD graph from the given process expression using the pLTS generated by the rules in Section 2. Then a sub-procedure *Dfsplit*, based on the depth-first search algorithm [3], is called. *Dfsplit* searches the nodes of the NPAD graph (which is a finite tree in this case) for finding those states that have more than one outgoing distributions. If such nodes exist, the NPAD graph can be split into a set of several small NPAD graphs, each of which is constructed by associating a different distribution with the state which is the first one to be found. Otherwise, the NPAD graph (which is a PAD graph) itself is returned. Consider the example in Figure 3. After the first time execution of *Dfsplit*, we find the left state in the second level of the tree in (a) has two distributions, and then the NPAD graph shown in (a) can be split into two NPAD graphs in (b).

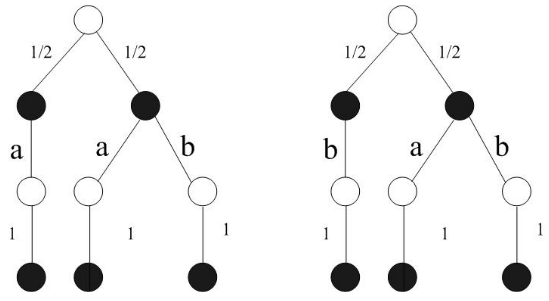
In presenting the following procedure, we recall the notion $m'_3([\![P + R]\!], [\![Q + R]\!])$ mentioned in Section 4. The procedure $semiHausd$ is implemented to calculate m'_3. Each time the procedure chooses a PAD graph in $[\![P]\!]$, it searches all the PAD graphs existing in $[\![Q]\!]$ to find such a PAD graph that makes $Distance$ as small as possible. The procedure repeatedly chooses each graph in $[\![P]\!]$ and obtain a set which contains all the smallest results. Then it picks up the largest one in this set. Finally, in order to get the final result $main\ loop$ of our algorithm executes $semiHausd$ twice.

Correctness of our algorithm lies in the fact that we use $semiHausd$ to calculate $m'_3([\![P]\!], [\![Q]\!])$ and $m'_3([\![Q]\!], [\![P]\!])$ respectively and then return the larger one. Therefore, we obtain $max(m'_3([\![P]\!], [\![Q]\!]), m'_3([\![Q]\!], [\![P]\!]))$ which is equal to $m([\![P]\!], [\![Q]\!])$ when our algorithm terminates.

The complexity of our algorithm is given by the follow-



(a)



(b)

Figure 3. Splitting an NPAD graph

ing proposition.

Proposition 5. *Given two NPAD graphs, each of which has n nodes, the time complexity of our algorithm is $\mathcal{O}(n^6 \log n)$.*

Proof. Consider an NPAD graph which has n nodes, the procedure $split$ will takes at most $\mathcal{O}(n^2)$ times to get a set $[\![P]\!]$ which includes at most n PAD graphs. Then the procedure $semiHausd$ is repeated at most n^2 times. Within one repeat of this procedure, the transshipment problem is called to calculate m_2.

As for the transshipment problem, it will take time $\mathcal{O}(n^2(m + n\log n)\log n)$ to find the result, where n and m denote the numbers of nodes and arcs respectively in the network [15]. According to our algorithm, the network is a complete graph and therefore it will take time $\mathcal{O}(n^4 \log n)$ to return the result. $\square$

7 Examples and Related Work

In this section, some examples are constructed only for the purpose of illustrating our algorithm we just discussed, and then we compare our metric with others.

Our algorithm calculates the distance between pCSP processes up to a prescribed discount factor c. Suppose $c = 1/2$, we calculate the distance of the following pairs of processes.

- Let

$$\begin{array}{rcl} P_1 & = & (a.0 \sqcap a.0)\,{}_{1/2}\!\oplus\, (b.0\,{}_{3/4}\!\oplus\, b.0) \\ P_2 & = & a.0\,{}_{1/3}\!\oplus\, (b.0 \Box b.0) \end{array}.$$

  It can be calculated that $m(P_1, P_2) = 1/6$.

- Let

$$\begin{array}{rcl} P_3 & = & a.0\,{}_{1/3}\!\oplus\, ((b.0 \sqcap c.0)\Box(d.0\,{}_{1/2}\!\oplus\, a.0)) \\ P_4 & = & (b.0 \sqcap c.0)\Box(d.0\,{}_{1/2}\!\oplus\, a.0). \end{array}$$

  It can be calculated that $m(P_3, P_4) = 1/3$.

We compare our metric with the one introduced by van Breugel in [18]. Our metric is a linear metric while theirs is a branching metric. Just as bisimulation implies trace equivalence, so branching metric is greater than or equal to the corresponding linear one. However, they restrict themselves to probabilistic transition systems without labels to simplify the definition, while our metric calculates the distance of two process terms directly.

We make a comparison with the metric introduced by Norman [14]. We argue that our metric is more accurate than theirs. Consider the following three processes:

$$\begin{array}{rcl} P & = & a.0\,{}_{1/2}\!\oplus\, b.0 \\ Q & = & d.a.0\,{}_{1/2}\!\oplus\, b.0 \\ R & = & d.0\,{}_{1/3}\!\oplus\, b.0. \end{array}$$

Based on the observations, we can infer that Q behaves more like R than P. This is captured by our metric, since P and R are $1/2$ apart whereas Q and R are $1/3$ apart. However, in the metric introduced by Norman both P, R and Q, R are $1/2$ apart.

Our metric concentrates on the finite probabilistic processes, while Norman's metric can deal with recursive probabilistic processes by means of approximations via truncations [13]. However, each approximation deals with finite trees, just as in our case, and no algorithm is considered in [13].

8 Conclusion and Future Work

In this paper we have proposed a metric for finite processes in a probabilistic extension of CSP. The kernel of our metric corresponds exactly to trace equivalence. Many process constructors has been shown to be nonexpansive with respect to our metric, except for parallel composition.

Making use of the transshipment problem, we have developed an algorithm that calculates the distance between finite probabilistic processes to a prescribed degree of accuracy in polynomial time. We have implemented our algorithm in a tool (see `http://basics.sjtu.edu.cn/~songlin/tool.html`). Given a discount factor and two process terms as input, the tool automatically calculates the distance between the two processes and produces it as output. All the examples in Section 7 have been checked by the tool.

As to the future work, we would like to extend our metric to capture the distance between probabilistic processes constructed with more operators so as to do some case studies. For example, we could write the specification of a security system as a perfect but impractical processes and its implementation as a practical process which is considered safe if it only differs from the specification with a negligible probability. We are now considering to integrate the method of [5] with our tool to measure the degree of anonymity for some security protocols.

Acknowledgments We thank Franck van Breugel for some helpful comments on our metrics.

References

[1] S. Brookes, C. Hoare, and A. Roscoe. A theory of CSP. *Journal of the ACM*, 31(3):560–599, 1984.

[2] V. Chvatal. *Linear programming*. W.H. Freeman and Co., New York/San Francisco, 1983.

[3] T. Cormen. *Introduction to Algorithms*. MIT Press, 2001.

[4] Y. Deng, T. Chothia, C. Palamidessi, and J. Pang. Metrics for action-labelled quantitative transition systems. *ENTCS*, 153(2):79–96, 2006.

[5] Y. Deng, J. Pang, and P. Wu. Measuring anonymity with relative entropy. In *Proceedings of the 4th International Workshop on Formal Aspects in Security and Trust*, volume 4691 of *LNCS*, pages 65–79. Springer, 2007.

[6] Y. Deng, R. van Glabbeek, M. Hennessy, C. Morgan, and C. Zhang. Characterising testing preorders for finite probabilistic processes. In *Proceedings of the 22nd Annual IEEE Symposium on Logic in Computer Science*, pages 313–325. IEEE Computer Society, 2007.

[7] Y. Deng, R. van Glabbeek, M. Hennessy, C. Morgan, and C. Zhang. Remarks on testing probabilistic processes. *ENTCS*, 172:359–397, 2007.

[8] J. Desharnais, V. Gupta, R. Jagadeesan, and P. Panangaden. Metrics for labelled Markov processes. *Theoretical Computer Science*, 318(3):323–354, 2004.

[9] J. Desharnais, R. Jagadeesan, V. Gupta, and P. Panangaden. The metric analogue of weak bisimulation for probabilistic processes. In *Proceedings of the 17th Annual IEEE Symposium on Logic in Computer Science*, pages 413–422. IEEE Computer Society, 2002.

[10] A. Giacalone, C. Jou, and S. Smolka. Algebraic reasoning for probabilistic concurrent systems. In *Proceedings of Working Conference on Programming Concepts and Methods*, volume 2 of *IFIP*, pages 453–459. Springer, 1990.

[11] C.-C. Jou and S. A. Smolka. Equivalence, congruences and complete axiomatisations for probabilistic processes. In *Proceedings of the 1st International Conference on Concurrency Theory*, volume 90 of *LNCS*, pages 234–248. Springer, 1990.

[12] L. Kantorovich. On the transfer of masses. *Dokl. Akad. Nauk. SSSR*, 37(7-8):227–229, 1942.

[13] M. Kwiatkowska and G. Norman. Metric denotational semantics for PEPA. Technical Report CSR-96-11, University of Birmingham, 1996.

[14] G. Norman. *Metric semantics for reactive probabilistic processes*. PhD thesis, University of Birmingham, 1997.

[15] J. Orlin, S. Plotkin, and É. Tardos. Polynomial dual network simplex algorithms. *Mathematical Programming*, 60(1):255–276, 1993.

[16] R. Segala. A compositional trace-based semantics for probabilistic automata. In *Proceedings of the 6th International Conference on Concurrency Theory*, volume 962 of *LNCS*, pages 234–248. Springer, 1995.

[17] R. Segala and N. Lynch. Probabilistic Simulations for Probabilistic Processes. *Nordic Journal of Computing*, 2(2):250–273, 1995.

[18] F. van Breugel and J. Worrell. An algorithm for quantitative verification of probabilistic transition systems. In *Proceedings of the 12th International Conference on Concurrency Theory*, volume 2154 of *LNCS*, pages 336–350. Springer, 2001.

[19] F. van Breugel and J. Worrell. Towards quantitative verification of probabilistic transition systems. In *Proceedings of the 28th International Colloquium on Automata, Languages and Programming*, volume 2076 of *LNCS*, pages 421–432. Springer, 2001.

A Pseudo-code

Algorithm 1. split(P)

```
$\mathcal{NP}(P) \leftarrow P$
$\mathscr{M} \leftarrow \mathcal{NP}(P)$, $\mathscr{N} \leftarrow \emptyset$
while $\mathscr{M} \neq \emptyset$ do
  $\mathcal{NP}(P')$ is a element of $\mathscr{M}$ and $\mathscr{M} = \mathscr{M}/\mathcal{NP}(P')$
  tempset ← Dfsplit($\mathcal{NP}(P')$)
  if tempset has only one element then
    $\mathscr{N} \leftarrow \mathscr{N} \cup$ tempset
  else
    $\mathscr{M} \leftarrow \mathscr{M} \cup$ tempset
  end if
end while
return $\mathscr{N}$
```

Algorithm 2. Distance($\mathcal{G}, \mathcal{G}', c$)

```
distance ← 0
$N \leftarrow |\mathscr{A}|$ -1
for k← 0 to N do
    for l← 0 to $N^2$+N-1 do
        matrix[k][l] is initiated
    end for
end for
for k←0 to N do
    demand[k]← $\mathbb{p}(u_k)$
end for
for l←0 to $N^2$+N-1 do
    cost[l] is initiated
end for
distance ← Transshipment(matrix,cost,demand)
return distance
```

Algorithm 3. semiHausd($[\![P]\!]$, $[\![Q]\!]$, c)

```
maxdistance ← 0
for all $\mathcal{G}$ in $[\![P]\!]$ do
    shortest ← $maximum$
    for all $\mathcal{G}'$ in $[\![Q]\!]$ do
        tempdis ← Distance($\mathcal{G}, \mathcal{G}', c$)
        if tempdis < shortest then
            shortest ← tempdis
        end if
    end for
    if shortest > maxdistance then
        maxdistance ← shortest
    end if
end for
return maxdistance
```

Algorithm 4. main loop

```
distance1 ← 0
distance2 ← 0
discount factor $c$ is initiated
$[\![P]\!]$ ← split(P)
$[\![Q]\!]$ ← split(Q)
distance1 ← semiHausd($[\![P]\!], [\![Q]\!], c$)
distance2 ← semiHausd($[\![Q]\!], [\![P]\!], c$)
return $\max$(distance1, distance2)
```