UNIVERSITY of York

This is a repository copy of *Probabilistic Graph Programming*.

White Rose Research Online URL for this paper:
https://eprints.whiterose.ac.uk/195045/

Version: Published Version

Proceedings Paper:
Plump, Detlef orcid.org/0000-0002-1148-822X, Atkinson, Timothy and Stepney, Susan orcid.org/0000-0003-3146-5401 (2017) Probabilistic Graph Programming. In: Corradini, Andrea, (ed.) Proceedings 8th International Workshop on Graph Computation Models (GCM 2017). , pp. 1-16.

Reuse
This article is distributed under the terms of the Creative Commons Attribution (CC BY) licence. This licence allows you to distribute, remix, tweak, and build upon the work, even commercially, as long as you credit the authors for the original work. More information and the full terms of the licence here: https://creativecommons.org/licenses/

Takedown
If you consider content in White Rose Research Online to be in breach of UK law, please notify us by emailing eprints@whiterose.ac.uk including the URL of the record and the reason for the withdrawal request.

White Rose university consortium
Universities of Leeds, Sheffield & York

eprints@whiterose.ac.uk
https://eprints.whiterose.ac.uk/

Probabilistic Graph Programming

Timothy Atkinson*, Detlef Plump, and Susan Stepney

Department of Computer Science, University of York, UK, YO10 5DD

Abstract. We introduce a notion of probability to the graph programming language GP 2 which resolves nondeterministic choices of graph transformation rules and their matches. With our programming model Probabilistic GP 2 (P-GP 2), rule and match decisions are assigned uniform distributions over their domains. In this paper, we present an implementation of P-GP 2 as an extension of an existing GP 2 compiler. As an example application, we analyse a (polynomial-time) nondeterministic vertex colouring program which may produce one of many possible colourings. The uniform implementation of P-GP 2 is shown, by sampling, to produce different colourings with different probabilities, allowing quantities such as expected colouring and likelihood of optimal colouring to be considered.

1 Introduction

As graph transformation becomes increasingly accessible to programmers through graph programming languages such as GP 2 [17], it becomes necessary to define programmatic behaviour over graph transformation systems which may not be deterministic in operation. Graph programming allows for the construction of multiple graph transformation systems, creating complex and useful transformations that can even be competitive with raw code at certain tasks [1]. However, the execution of a program can be nondeterministic, where the result graph cannot be predicted and different valid implementations of GP 2 can perform different executions of the same program.

Nondeterministic programs may be of interest to a number of current research topics. Notably, cryptography and machine learning are areas where probabilistic decision making is frequently used. By extending graph programming to the probabilistic domain it is hoped that the intuition and formalism that graph transformation offers might aid development in these areas. The authors of this work have particular interest in the application of graph transformation to evolutionary computation, where probabilistic transformations are used to solve optimisation and black box problems.

In this paper we discuss a programming model, probabilistic GP 2 (P-GP 2), as a means of resolving nondeterminism in GP 2. P-GP 2 assigns a uniform probability distribution to each of GP 2's nondeterministic rule-choice and matching

* Supported by a Doctoral Training Grant from the Engineering and Physical Sciences Research Council (EPSRC) in the UK.

decisions, a notion understood through probability operated graph transformation systems. This probabilistic approach is shown to induce a Markov chain, as the probabilities for state transitions are fixed.

Implemented as an extension of an existing C-based compiler for GP 2 [3], we show how this probabilistic refinement of GP 2 defines predictable behaviour which can be sampled. For example, we identify a simple GP 2 vertex colouring program which has nondeterministic execution. Executing this algorithm with P-GP 2, it is shown that the ambiguity in the result is resolved and the behaviour of the program can be understood in terms of quantities such as the expected colouring and the likelihood of optimal colouring for a given input.

There are two related approaches to the probabilistic operation of graph transformation systems; stochastic and probabilistic. The former describes stochastic graph transformation systems (SGTS), where each rule in a graph transformation system's rule-set is associated with a real positive value, known as a rule's application rate [8,9]. The probability of each match for a given rule occurring in continuous time is typically described according to an exponential distribution parameterised by the rule's application rate. An implication of using SGTSs is that the probability of a rule from the rule-set being applied in any given step is dependent on the number of matches for that rule. SGTSs have been generalised to a model where each match for each rule, referred to as an *event*, is associated with some continuous probability distribution, inducing generalised semi-Markov schemes describing the operation of the entire system [11,21,10].

Probabilistic graph transformation systems (PGTS) walk a middle ground between classical graph transformation's freedom of choice and more probabilistic notions. PGTSs are graph transformation systems where the choice of rule and match are nondeterministic decisions but rules have different possible executions (based on a common left hand side) which occur with different probabilities [13]. This mixture of nondeterminism and probabilities over discrete space is shown to induce Markov decision processes. The Markovian processes induced by the stochastic and probabilistic notions are distinct; PGTSs are based on discrete, rather than continuous, time and maintain a degree of nondeterminism. Both of these approaches are discussed in relation to P-GP 2 in Section 5.

The rest of this paper is structured as follows. In Section 2, we introduce the graph programming language GP 2, and in Section 3 we describe an example nondeterminstic GP program and demonstrate the resolution of this program's nondeterminism using uniform distributions over rule matches. In Section 4 we define P-GP 2 and discuss its properties, and in Section 5 we relate this concept to existing work. Finally, we conclude in Section 6.

2 Graph Programming with GP 2

We explain some features of the graph transformation language GP 2 that are relevant for this paper. The original language definition, including an operational semantics, is given in [17]; an updated version can be found in [1]. There are

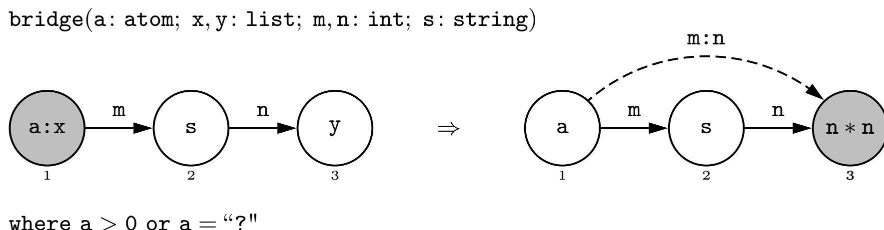

Fig. 1. Declaration of a conditional rule, `bridge`.

currently two implementations of GP 2, a compiler generating C code [3] and an interpreter for exploring the language's nondeterminism [2].

GP 2 programs transform input graphs into output graphs, where graphs are labelled and directed and may contain parallel edges and loops.

Labels consist of an expression and an optional mark (explained below). Expressions are of type `int`, `char`, `string`, `atom` or `list`, where `atom` is the union of `int` and `string`, and `list` is the type of a (possibly empty) list of atoms. Lists of length one are equated with their entries and hence every expression can be considered as a list. Types form a hierarchy with the most general type `list` which is a supertype of `atom` which in turn has subtypes `int` and `string`.

The concatenation of two lists x and y is written $x{:}y$. Character strings are enclosed in double quotes. Composite arithmetic expressions such as `n ∗ n` must not occur in the left-hand graph, and all variables occurring in the right-hand graph or the condition must also occur in the left-hand graph.

Besides carrying list expressions, nodes and edges can be *marked*. In Figure 1, the outermost nodes are marked by a grey shading and the dashed arrow between nodes 1 and 3 in the right-hand graph is a marked edge. Marks are convenient to highlight items in input or output graphs, and to record visited items during a graph traversal.

The principal programming constructs in GP 2 are conditional graph transformation rules labelled with expressions. For example, Figure 1 shows the declaration of the rule `bridge`, which has six formal parameters of various types, a left-hand graph and a right-hand graph that are specified graphically, and a textual condition starting with the keyword `where`. The small numbers attached to nodes are identifiers, all other text in the graphs are labels.

Rules operate on *host graphs* which are labelled with constant values (lists containing integer and string constants). We write $\mathcal{G}$ for the set of all host graphs. Applying a rule $L \Rightarrow R$ to a host graph G works as follows: (1) Find an injective pre-morphism $g\colon L \to G$. (A pre-morphism maps nodes to nodes and edges to edges such that each edge keeps its source and target node.) (2) Check if there is an assignment α of values to variables that turns g into a graph morphism $g\colon L^\alpha \to G$, where L^α results from applying α to the variables in L's labels. (A pre-morphism is a graph morphism if it preserves labels.) (3) Check that g satisfies the dangling condition and the rule's application condition (see below).

(4) Replace the subgraph $g(L^\alpha)$ of G with R^α as follows: numbered nodes of L^α stay in place (possibly relabelled), edges and unnumbered nodes are deleted, and edges and unnumbered nodes of R^α are inserted.

In this construction, the *dangling condition* requires that nodes in $g(L^\alpha)$ that are unnumbered in L^α (and hence should be deleted) must not be incident with edges outside $g(L^\alpha)$. The rule's condition is evaluated after each variable x has been replaced with $\alpha(x)$. For example, the condition of Figure 1 requires that the list label of node $g(\mathtt{1})$ starts with a positive integer or with the string "`?`".

If clauses (2) and (3) above are satisfied, we call the pair (r, g) a *match* of r in G. We write $G \Rightarrow_{r,g} H$ if H is isomorphic to the graph constructed in (4). The GP 2 syntax imposes restrictions on the left-hand graph L such that the pre-morphism $g\colon L \to G$ induces at most one assignment α that turns g into a graph morphism. Hence the following holds.

Proposition 1. *Given a match (r, g) of r in G, there exists up to isomorphism exactly one graph H such that $G \Rightarrow_{r,g} H$.*

A GP 2 program consists of declarations of conditional rules and procedures, and exactly one declaration of a main command sequence following the key word `Main`. Commands include calls to individual rules and to rule sets $\{r_1, \ldots, r_n\}$. The latter call nondeterministically picks a rule r_i that has a match in the host graph and applies it. The rule set call *fails* if none of the rules has a match.

We omit the description of branching commands and procedures as they are not needed in this paper. A loop command $P!$ executes the body P repeatedly until it fails. When this is the case, $P!$ terminates with the graph on which the body was entered for the last time. The `or` command nondeterministically chooses between two program branches; this source of nondeterminism is only briefly discussed, in Section 4.2, due to its simplicity.

In general, the execution of a program on a host graph may result in different graphs, fail, or diverge.

3 Case Study: Vertex Colouring

In this section we discuss a very simple nondeterministic vertex colouring program VC (taken from [18]). Computing a vertex colouring that uses the minimal number of colours is a NP-complete problem [20], the program VC only guarantees to compute *some* colouring but does this in polynomial time. In Section 3.3 we discuss the behaviour of VC under P-GP 2 on members of a problem set, grid graphs, that have known optimal colourings. This gives value to a probabilistic analysis of VC that was previously not possible.

3.1 Grid Graphs

In a grid graph, nodes are organized in a square lattice. We give direction to grid graphs by allocating one node as a source and all edges directed outwards of that source. Figure 2 shows a 5×3 grid graph with node 1 as its source. Let

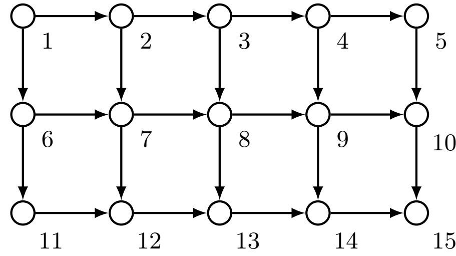

Fig. 2. $GG_{5,3}$: a 5×3 grid graph

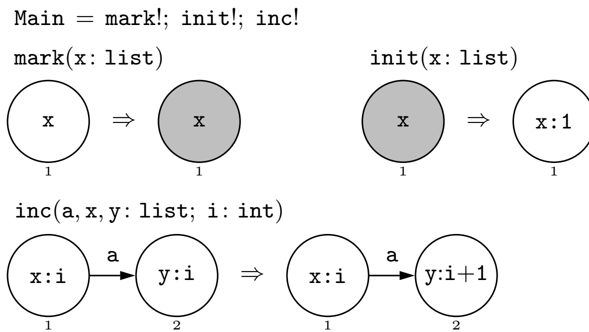

Fig. 3. The program VC

GG be the family of all unlabelled directed grid graphs and $GG_{x,y}$ specifically refer to a $x \times y$ unlabelled directed grid graph.

In this case study, we discuss the likelihood of VC producing an optimal colouring over GG in terms of parameters x and y. We choose GG as a motivating example as each of its members has a known optimal colouring. Using 2 colours, a grid graph can be coloured in a checkerboard fashion, and GG is therefore a family of bipartite graphs.

3.2 Vertex Colouring Program VC

Figure 3 shows our graph colouring program VC. The colour assigned to a particular node is an integer which is appended to the node's existing label. The first part of the program, `mark!;init!`, is deterministic, assigning to each node the colour `1`. The second part of the program, the loop `inc!`, is terminating but highly nondeterministic, matching adjacent pairs of nodes that are identically coloured and incrementing the colour of the target node of the matched connecting edge.

The time complexity of the initial part is quadratic in the number of host graph nodes; both `mark` and `init` are applied to each node once and finding a match for either rule requires a single search over all nodes. It can be shown that

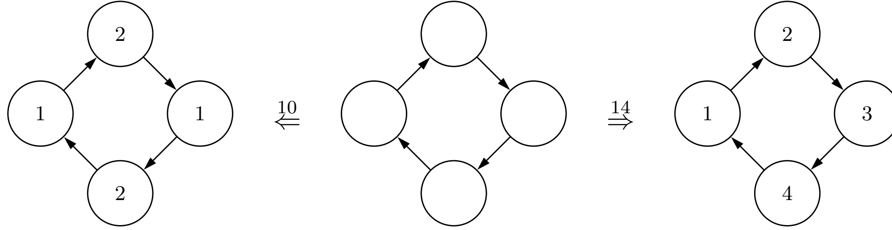

Fig. 4. Different results from executing VC

the number of `inc` applications is quadratic in the number of host graph nodes [18]. Moreover, the compiled GP 2 code will find a match of `inc` in linear time by searching once over all edges in the host graph. Therefore the run time of the loop `inc!` is cubic in the size of the host graph and hence VC's overall time complexity is also cubic.

Figure 4 shows two executions of VC on a small host graph. Whereas the left execution produces an optimal colouring with 10 rule applications, the right execution returns the worst colouring after 14 steps. Because of the nondeterminism, different (deterministic) implementations of GP 2 could produce entirely different results for VC, in accordance with the language specification.

3.3 Behaviour with Uniform Matching

To address VC's nondeterminism, we ran the algorithm using a uniform distribution over `inc`'s matches to make the behaviour of the program correspond to a clearly defined probability distribution over result graphs. These uniform distributions over matches are a feature of P-GP 2, discussed in detail in Section 4.

While such an implementation of GP 2 can be sampled, exhaustive sampling is expensive; this is an issue we seek to move away from in future work. In the interim, it suffices to describe the behaviour of a program on a given input to an arbitrary degree of detail, here chosen to be when 5% or less of the behaviour is unknown. This can be achieved by sampling more than 95% of the program's behaviour on a given input.

We study the *likelihood of optimal colouring* for a set of inputs, given as the cumulated probability of samples producing an optimal colouring, which in this scenario corresponds to samples producing a 2 colouring. Table 1 shows the observed behaviour of VC over grid graphs with width in the integer interval $[1, 3]$ and height in $[1, 5]$. Each result is given as a pair (a, b) where a describes the degree to which the program was sampled and b describes the summed probability of samples for that input which returned 2 coloured result graphs.

As an observation, the likelihood of generating a 2 colouring for a grid graph appears greatly reduced as the graph grows in width and height. Figure 5 shows the worst case (assuming all remaining samples do not produce optimal colourings) likelihood of optimal colouring of grid graph classes $GG_{1,x}$, $GG_{2,x}$ and $GG_{3,x}$ from Table 1, each indicated by a separate series.

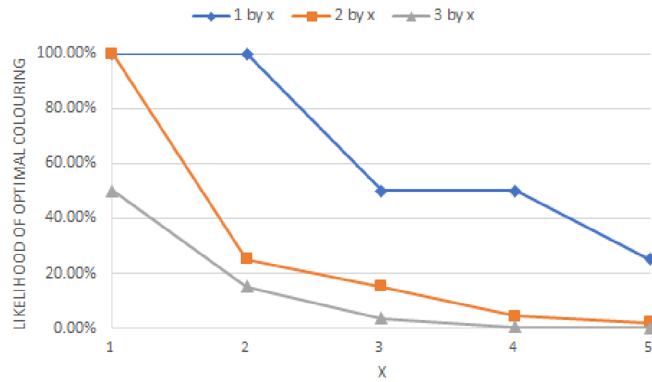

Fig. 5. Worst case scenario described by VC's observed behaviour. Each series corresponds to a group of grid graphs of a given width, 1 2 or 3. Each data point in a series corresponds to the worst case likelihood of a particular graph of that series' width and x height being optimally coloured by VC.

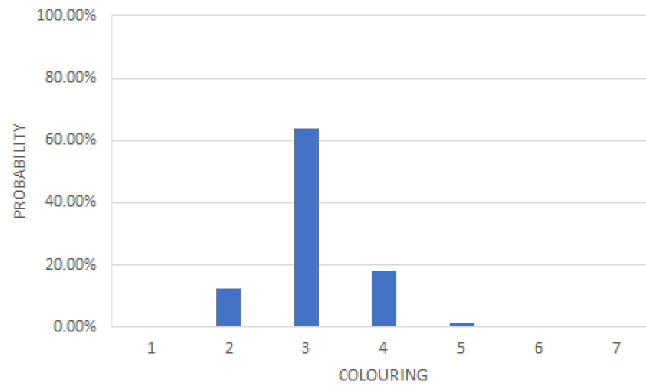

Fig. 6. Probability of different colourings when executing VC on $GG_{1,7}$. Each column shows the summed probability of observed samples producing each possible colouring.

		width		
		1	2	3
height	1	(100%, 100%)	(100%, 100%)	(100%, 50%)
	2	(100%, 100%)	(100%, 25%)	(96.19%, 15.24%)
	3	(100%, 50%)	(96.19%, 15.24%)	(95.11%, 3.55%)
	4	(100%, 50%)	(95.13%, 4.29%)	(95.00%, 0.43%)
	5	(100%, 25%)	(95.01%, 2.00%)	(95.00%, 0.01%)

Table 1. Results from sampling members of GG for derivations producing optimal colourings. The first value in an entry indicates the proportion of the space sampled, the second value indicates the proportion of samples producing optimal colourings.

It is also possible to study the behaviour of VC in depth for particular inputs. The *expected colouring* of an input is the average colour we expect VC to produce, corresponding to the notion of expected value more generally. This value is calculated as the weighted sum of colourings produced by VC according to their probability. We take the input $GG_{1,7}$ as such an example and measure the probability of each of its possible colourings when sampling 95% of the distribution of its behavior. Figure 6 shows the results of this sampling. From our observations, the expected colouring is 3.08. Assuming the best case (all remaining samples are 2 coloured), the expected colouring is 3.03 whereas assuming the worst case (all remaining samples are 7 coloured) the expected colouring is 3.27.

We therefore claim to have resolved the nondeterminism in VC's operation; the behaviour of VC can be sampled when using P-GP 2.

4 Probabilistic GP 2 (P-GP 2)

In this section we describe the programming model probabilistic GP 2 (P-GP 2). The model described here has been implemented as an extension of the existing C-based compiler [1].

P-GP 2 is a refinement of GP 2 where nondeterministic choices are decided according to probability distributions, rather than arbitrary determinism. Consider a call to a rule-set $\mathcal{R}$ with host graph G. The expected outcome of this call is $G \Rightarrow_r H$, where some rule $r \in \mathcal{R}$ is applied to G producing graph H. Additionally, if no rule $r \in \mathcal{R}$ has a valid match in G, the rule-set call fails, acting as a deterministic operation.

A rule-set call's execution requires a decision over the set of possible matches for all $r \in \mathcal{R}$ in the host graph G, a notion drawn from generalised stochastic graph transformation systems [21]. Each match for some rule r is identified using a rule-match pair (r, g), where g is a pre-morphism, rather than a pre-morphism alone to retain information about which rule induces a certain pre-morphism. We refer to this decision space as the *projection* of $\mathcal{R}$ onto a graph G, denoted $G^{\mathcal{R}}$:

$$G^{\mathcal{R}} = \{(r, g) \mid r \in \mathcal{R} \text{ and } G \Rightarrow_{r,g} H \text{ for some graph } H\} \quad (1)$$

A rule-set $\mathcal{R}$ has a corresponding set of all possible rule-match pairs over all possible host graphs $\mathcal{G}$, denoted as $\mathcal{X}^{\mathcal{R}}$[1]:

$$\mathcal{X}^{\mathcal{R}} = \bigcup_{G \in \mathcal{G}} G^{\mathcal{R}} \quad (2)$$

We associate a function $P_{\mathcal{R}}$ with each rule-set $\mathcal{R}$ used in a program such that when calling $\mathcal{R}$ on any possible host graph G, $P_{\mathcal{R}}$ provides a probability distribution over $G^{\mathcal{R}}$. Let $[0.0, 1.0]$ denote the real-valued interval between 0 and 1. The function $P_{\mathcal{R}}$ for a given rule-set $\mathcal{R}$ has domain and co-domain:

$$P_{\mathcal{R}} : \mathcal{X}^{\mathcal{R}} \to [0.0, 1.0] \quad (3)$$

For a given projection $G^{\mathcal{R}}$, the probability distribution assigned to it must sum to 1:

$$\sum_{\forall x \in G^{\mathcal{R}}} P_{\mathcal{R}}(x) = 1 \text{ if } G^{\mathcal{R}} \neq \emptyset \quad (4)$$

We use $P_{\mathcal{R}}$ to decide which member $(r, g) \in G^{\mathcal{R}}$ to execute when calling rule-set $\mathcal{R}$ on some host graph G. In the case where $\mathcal{G}^{\mathcal{R}}$ is empty and there are no applicable rules, the application of $\mathcal{R}$ to G becomes a deterministic operation returning `fail`.

For simplicity, in this paper we treat rule choice and match choice as separate decisions. A rule r is chosen with uniform probability from the set of rules with valid matches in the host graph G, and a match is then chosen with uniform probability from the set of possible matches for r in G. This approach has two practical benefits facilitating the extension of GP 2 to the probabilistic domain; (1) rule choice and match choice are already treated as separate decisions in existing implementations [1,2], and (2) uniform distributions have no additional parameters so require no new syntax. This approach minimizes change to GP 2's implementation and leaves the language specification unchanged.

For a rule-set $\mathcal{R}$, $P_{\mathcal{R}}$ is defined for any pair $(r, g) \in \mathcal{X}^{\mathcal{R}}$, where $r = (L \leftarrow K \rightarrow R, c)$ and $g : L \rightarrow G$ describes a match of r in some graph G, as:

$$P_{\mathcal{R}}(r, g) = P_{\mathcal{R}}^{rule}(r, g) \times P_{\mathcal{R}}^{match}(r, g) \quad (5)$$

Here the probability of choosing a pair $(r, g) \in G^{\mathcal{R}}$ is equal to the probability of choosing the rule, $P_{\mathcal{R}}^{rule}(r, g)$, multiplied by the probability of choosing the match if the rule is chosen, $P_{\mathcal{R}}^{match}(r, g)$. A uniform distribution is assigned to the choice of rule, assigning equal probability to each member of $\mathcal{R}^G$, the set of rules from $\mathcal{R}$ that have matches in G:

[1] This set corresponds to the set of events, $\mathcal{E}^{\mathcal{R}}$, used in [21], but we avoid that terminology as we are outside of the continuous modelling domain.

$$\mathcal{R}^G = \{r \in \mathcal{R} \mid G \Rightarrow_{r,f} H \text{ for some pre-morphism } f \text{ and host graph } H\} \quad (6)$$

$$P_{\mathcal{R}}^{rule}(r, g) = \frac{1}{|\mathcal{R}^G|} \quad (7)$$

Note that for (r, g) to exist, it must be true that $r \in \mathcal{R}^G$. Let G^r denote the projection of the single rule r onto host graph G. Then the probability of choosing the specific pre-morphism g once rule r has been chosen is defined:

$$P_{\mathcal{R}}^{match}(r, g) = \frac{1}{|G^r|} \quad (8)$$

This gives an overall formula for $P_{\mathcal{R}}$ for any rule $r = (L \leftarrow K \rightarrow R, c)$ and associated pre-morphism $g : L \rightarrow G$ in some host graph G as:

$$P_{\mathcal{R}}(r, g) = \frac{1}{|\mathcal{R}^G|} \times \frac{1}{|G^r|} \quad (9)$$

In Subsection 4.1 we describe probability operated graph transformation systems, the intuition behind P-GP 2, which we show to induce Markov chains. In Subsection 4.2 we reason that these POGTSs cause P-GP 2 programs executed on individual inputs to induce Markov chains.

4.1 Probability Operated Graph Transformation Systems (POGTS)

In this section we describe the intuition behind using probability distributions to operate the execution of a given graph transformation system in discrete time, calling this model *probability operated graph transformation systems* (POGTS). This intuition is the basis for the programming model P-GP 2; a single rule-set call $\mathcal{R}$ is the execution of a single step of a POGTS, while calling a rule-set call as long as possible, $\mathcal{R}!$, is the execution of a POGTS until termination.

The main concept can be understood as associating a conventional graph transformation system with a function P such that when applying $\mathcal{R}$ to some possible graph G, there is an associated probability distribution over $G^{\mathcal{R}}$ provided by P. A POGTS is theoretically capable of describing a unique distribution for every possible graph, such that even if two graphs have similar topologies, their assigned distributions may be entirely different.

Definition 1. ***(Probability Operated Graph Transformation System).*** *A $POGTS = (\mathcal{R}, P)$ is a finite set of rules $\mathcal{R}$ and a function P such that when transforming any graph G, P associates a fixed probability with each pair $(r, g) \in G^{\mathcal{R}}$:*

$$P : \mathcal{X}^{\mathcal{R}} \rightarrow [0.0, 1.0] \tag{10}$$

$$\sum_{(r,g) \in G^{\mathcal{R}}} P(r, g) = 1 \textit{ if } G^{\mathcal{R}} \neq \emptyset \tag{11}$$

Each possible transformation, $G \Rightarrow_{r,g} H$, is executed with probability $P(r, g)$.

We assume a discrete time model for POGTSs as we are only concerned with the step-wise operation of a graph transformation system, rather than a specific modelling domain.

Each POGTS applied to a graph induces a Markov chain. A Markov chain is a model in probability theory where there are transitions between states in a countable set S occurring with fixed probabilities [19,16]. This is viewed as a Markov process, see Definition 2, over a discrete, countable state space.

Definition 2. ***(Markov process)[19,16].*** *A Markov process is a stochastic process $\mathbb{X} = (X_0, X_1, X_2, ...X_n)$ consisting of a sequence of random variables where for each random variable X_i at time i, all future states are conditionally dependent on the current state and independent from previous states:*

$$Pr(X_{i+1} = x \mid X_0 = x_0, X_1 = x_1, X_2 = x_2, ..., X_i = x_i) = Pr(X_{i+1} = x \mid X_i = x_i) \tag{12}$$

Fixed probabilities mean that the probability of transitioning from one state to another depends only on the current state. The transition probabilities can be represented as a $|S| \times |S|$ transition matrix Q where for any two states $s, s' \in S$, $Q(s, s')$ is the probability of transitioning from state s to state s'. The behavior of the process can then be simulated by repeatedly multiplying initial distribution X_0, a vector of size $|S|$ describing a probability distribution of the process's initial state, by Q. After n transitions (time steps) this produces vector X_n describing the probability of being in a state $s \in S$ as $X_n(s)$. If S is countable but infinite, there may be no natural representation for Q.

Definition 3. ***(Markov chain) [19,16].*** *A Markov chain is a Markov process $\mathbb{X} = (X_0, X_1, X_2, ...X_n)$ on a countable state space S, such that each random variable X_i at time i is a probability distribution over S.*

For a POGTS $(\mathcal{R}, P)$ applied to graph G, the induced Markov chain's state space S is every graph reachable by repeatedly applying $\mathcal{R}$ to G:

$$S = \{H \mid G \Rightarrow^* H\} \tag{13}$$

For any POGTS and input graph G, the implied state space must be a subset of the set of all possible host graphs: $S \subset \mathcal{G}$. As $\mathcal{G}$ is countable, it entails that S must always be countable. The induced transition matrix Q is defined according

the possible transitions between pairs of graphs $A, B \in S$ and their associated fixed probabilities given by P:

$$Q(A, B) = \sum_{(r,g) \in A^{\mathcal{R}} \mid A \Rightarrow_{r,g} B} P(r, g) \tag{14}$$

Informally speaking, the transition matrix entry for the transition between graphs A and B is the total probability of A being transformed into B in a single step by the POGTS using any of the matches in $A^{\mathcal{R}}$.

The initial distribution X_0 is a trivial case; the probability of being in initial state G, the input graph, is 1. This means that the initial distribution can be defined, for any graph $G' \in S$, as:

$$X_0[G'] = \begin{cases} 1 & \text{if } G' = G \\ 0 & \text{otherwise} \end{cases} \tag{15}$$

In special cases, it may be possible to consider transition matrix Q explicitly for a POGTS and find probabilities of its resultant graph accordingly, but more generally an input graph may not be known until a program using POGTSs is executed preventing pre-computation of state space S and therefore Q. In this case, a step-wise execution of some POGTS to produce a result graph can be seen as sampling from its induced Markov chain. The execution of a single rule-set call in P-GP 2 corresponds to a single step of the induced Markov chain of the corresponding POGTS, whereas the as-long-as-possible call $\mathcal{R}!$ corresponds to steps of the induced Markov chain until reaching some absorbing state (see [16] for more information).

Probability operated graph transformation systems are compared to related work in Section 5.

4.2 P-GP 2 as a Markov chain

A P-GP 2 program executed on some input graph induces a Markov chain constructed from component Markov chains. This claim depends on the knowledge that all GP 2 operations excepting rule-set calls and `or` commands are deterministic in operation, as can be seen in the operational semantics of GP 2 [17]. The `or` command is assigned simple probabilistic behavior, both here and in [1], where the probability of taking either branch is fixed to 50%.

A P-GP 2 program's execution can be discussed in terms of its operational semantics over configurations, where a configuration (Q, G) is a sub-program Q to be executed, and a host graph G. As a program consists of finitely many commands, there are finitely many possible sub-programs Q. $G \in \mathcal{G}$, the countable set of possible host graphs, and therefore the set of reachable configurations for a program must be countable. Each rule-set call treated as a POGTS induces a Markov chain as described in Section 4.1. Every P-GP 2 program executed on some input graph therefore describes a scenario where operations are either deterministic, execute simple fixed probabilistic behaviors (`or` commands) or

induce Markov chains, with each operation leading to a new configuration or termination. Viewing configurations as the state space of a P-GP 2 program, the program is constrained to a countable state space where each probabilistic decision depends only on the current state; the P-GP 2 program itself induces a Markov chain. This logic extends to branching commands and Procedure calls, where nondeterminism is ultimately resolved by probabilistic rule-set calls and `or` commands.

It is possible to sample from the Markov chain underlying an entire P-GP 2 program to build a partial distribution of its behaviour. This can be done by repeatedly executing a P-GP 2 program on a given input and recording the resultant graph and probability of its derivation. By ignoring duplicate derivations with identical matching morphisms and `or` branching choices, a partial distribution can be built to an intended degree of completeness. This technique is used to produce the results given in the case study in Section 3.2; the probability of producing a graph is given by the summed probability of observed derivations producing that graph.

5 Related Work

Comparing the stochastic and probability operated models, the first distinction is the choice of time model. The former are concerned with real time modelling problems, and therefore use a continuous time model, whereas the latter are concerned with operating the discrete steps of a graph transformation system and therefore use a discrete time model. The conventional stochastic graph transformation model uses function $F : \mathcal{R} \to \mathbb{Z}^+$ to assign to each rule an application rate which induces some continuous time distribution [8]. This means that the discrete decision processes of POGTSs cannot be universally simulated in SGTSs; the probability of a rule being applied in the stochastic setting is dependent on the number of matches for that rule, making uniform distributions over rule choice such as those described in Section 4 impossible.

However, for generalised stochastic systems [21], function $F : \mathcal{X}^{\mathcal{R}} \to [\mathbb{R} \to [0.0, 1.0]]$ assigns each possible pair (r, g) a continuous time distribution. This is clearly related to probability operated graph transformation systems, and it may be possible to simulate the distribution over produced result graphs for each approach in the other. However, these two approaches are philosophically distinct, using different models for time to address different problems: POGTSs are used to facilitate algorithms on graphs whereas SGTSs are used for real time modelling.

Probabilistic graph transformation systems [13] share a discrete time model with the probability operated setting. The choice of rule and match in the probabilistic setting is nondeterminstic, so while an individual execution in the probability operated setting can be described by nondeterministic choices in the probabilistic setting, PGTSs exhibit the same ambiguity in execution characteristic of classical graph transformation systems. It may be possible to simulate probabilistic rule executions in the probability operated setting by using rules

Model	Deciding Rule & Match Choice	Time Model	Probabilistic Rule Execution
Classical Graph Transformation	Nondeterministic	—	No
Probabilistic Graph Transformation	Nondeterministic	D	Yes
Stochastic Graph Transformation	Probabilistic	C	No
Probability Operated Graph Transformation	Probabilistic	D	No

Table 2. Different approaches to decision making in graph transformation. D indicates discrete time, C indicates continuous time.

with common left hand sides, although whether this is universally true remains to be seen. Table 2 explains the distinction between conventional graph transformation, stochastic graph transformation, probabilistic graph transformation and the probability operated graph transformation model used in this paper.

As this paper deals with the graph programming language GP 2, it is natural to examine how probabilistic decisions are made in mainstream probabilistic programming languages. Typically, a probabilistic programming language is primarily capable of drawing values from random distributions and conditioning variables on observations [7]. There are a number of dedicated probabilistic languages spanning paradigms, for example, the functional language BUGS [6], C# based Infer.NET [15] and SQL variant PSQL [4]. A key notion contained within this field is that probability distributions are defined and conditioned within language constructs, rather than constructing procedures with pseudo random number generators to achieve probabilistic behaviour as one might in conventional languages. For example, the language *Picture* [14] elegantly demonstrates the expressiveness brought by dedicated probabilistic programming; competing with highly specialised scene recognition programs using concise programs described in a new language. There is a parallel here; P-GP 2 allows the programmer to describe a probability distribution over result graphs conditioned on the input graph which can be sampled, although whether more advanced notions such as inference are relevant remains to be seen.

6 Conclusion and Future Work

In this paper we have identified a nondeterministic vertex colouring algorithm and proposed a programming model, P-GP 2, as a means of overcoming this nondeterminism and executing GP 2 programs consistently. Using uniform distributions over matches, we have measured quantities such as expected colouring

and likelihood of optimal colouring for a set of input graphs. This allows quantitative evaluation of the vertex colouring algorithm under P-GP 2 that was previously unavailable.

Additionally, we have described the underlying decision process used to guide P-GP 2 as probability operated graph transformation systems. These systems induce Markov chains, transitioning over a countable state space with fixed probabilities. This notion is used to justify the treatment of a P-GP 2 executing on some input graph as a Markov chain for sampling.

We note a number of directions in which P-GP 2 can be extended. Firstly, the ability of a programmer to define custom distributions over a projection $G^{\mathcal{R}}$ may be useful for implementing nondeterministic algorithms. For example, there are Genetic Algorithms which have both crossover and mutation operators which are applied with different probabilities [5]. Treating both operators as rules, a programmer might even wish to optimise their algorithm for a specific domain by tuning these probabilities throughout execution [22]. The role of varying probabilities in match choice remains unclear, although the use of rule schemata in GP 2 potentially allows quantitative distinction between matches according to their assignments. It is also appealing to allow user driven choice in parts of the execution of a GP 2 program as this would both allow full simulation of probabilistic graph transformation systems and accommodate for user driven nondeterministic algorithms such as human based genetic algorithms [12].

At the time of writing, we leave it open as to how GP 2 constructs should be extended to allow the programmer to define different distributions across their program.

References

1. Bak, C.: GP 2: Efficient Implementation of a Graph Programming Language. Ph.D. thesis, Department of Computer Science, University of York (2015), http://etheses.whiterose.ac.uk/12586/
2. Bak, C., Faulkner, G., Plump, D., Runciman, C.: A reference interpreter for the graph programming language GP 2. In: Proc. Graphs as Models (GaM 2015). Electronic Proceedings in Theoretical Computer Science, vol. 181, pp. 48–64 (2015), doi:10.4204/EPTCS.181
3. Bak, C., Plump, D.: Compiling graph programs to C. In: Proc. International Conference on Graph Transformation (ICGT 2016). LNCS, vol. 9761, pp. 102–117. Springer (2016), doi:10.1007/978-3-319-40530-8_7
4. Dey, D., Sarkar, S.: PSQL: A query language for probabilistic relational data. Data & Knowledge Engineering 28(1), 107–120 (1998), doi:10.1016/S0169-023X(98)00015-9
5. Eiben, A.E., Smit, S.K.: Parameter tuning for configuring and analyzing evolutionary algorithms. Swarm and Evolutionary Computation 1(1), 19–31 (2011), doi:10.1016/j.swevo.2011.02.001
6. Gilks, W.R., Thomas, A., Spiegelhalter, D.J.: A language and program for complex bayesian modelling. Journal of the Royal Statistical Society. Series D (The Statistician) 43(1), 169–177 (1994), http://www.jstor.org/stable/2348941

7. Gordon, A.D., Henzinger, T.A., Nori, A.V., Rajamani, S.K.: Probabilistic programming. In: Proc. Future of Software Engineering (FOSE 2014). pp. 167–181. ACM (2014), doi:10.1145/2593882.2593900
8. Heckel, R.: Stochastic analysis of graph transformation systems: A case study in P2P networks. In: Proc. Theoretical Aspects of Computing (ICTAC 2005), Second International Colloquium. LNCS, vol. 3722, pp. 53–69. Springer (2005), doi:10.1007/11560647_4
9. Heckel, R., Lajios, G., Menge, S.: Stochastic graph transformation systems. Fundamenta Informaticae 74(1), 63–84 (2006), http://content.iospress.com/articles/fundamenta-informaticae/fi74-1-04
10. Heckel, R., Torrini, P.: Stochastic modelling and simulation of mobile systems. In: Graph Transformations and Model-Driven Engineering - Essays Dedicated to Manfred Nagl on the Occasion of his 65th Birthday. LNCS, vol. 5765, pp. 87–101. Springer (2010)
11. Khan, A., Heckel, R., Torrini, P., Ráth, I.: Model-based stochastic simulation of P2P voip using graph transformation system. In: Proc. International Conference on Analytical and Stochastic Modeling Techniques and Applications (ASMTA 2010). LNCS, vol. 6148, pp. 204–217. Springer (2010), doi:10.1007/978-3-642-13568-2_15
12. Kosorukoff, A.: Human based genetic algorithm. In: 2001 IEEE International Conference on Systems, Man and Cybernetics. e-Systems and e-Man for Cybernetics in Cyberspace (Cat.No.01CH37236). vol. 5, pp. 3464–3469 vol.5 (2001)
13. Krause, C., Giese, H.: Probabilistic graph transformation systems. In: Proc. International Conference on Graph Transformations (ICGT 2012). LNCS, vol. 7562, pp. 311–325. Springer (2012), doi:10.1007/978-3-642-33654-6_21
14. Kulkarni, T.D., Kohli, P., Tenenbaum, J.B., Mansinghka, V.K.: Picture: A probabilistic programming language for scene perception. In: Proc. IEEE Conference on Computer Vision and Pattern Recognition, CVPR. pp. 4390–4399. IEEE Computer Society (2015), doi:10.1109/CVPR.2015.7299068
15. Minka, T., Winn, J., Guiver, J., Webster, S., Zaykov, Y., Yangel, B., Spengler, A., Bronskill, J.: Infer.NET 2.6 (2014), http://research.microsoft.com/infernet, microsoft Research Cambridge.
16. Norris, J.R.: Markov chains. Cambridge series in statistical and probabilistic mathematics, Cambridge University Press (1998)
17. Plump, D.: The design of GP 2. In: Proc. International Workshop on Reduction Strategies in Rewriting and Programming (WRS 2011). EPTCS, vol. 82, pp. 1–16 (2011), doi:10.4204/EPTCS.82.1
18. Plump, D.: Reasoning about graph programs. In: Proc. Computing with Terms and Graphs (TERMGRAPH 2016). Electronic Proceedings in Theoretical Computer Science, vol. 225, pp. 35–44 (2016), doi:10.4204/EPTCS.225.6
19. Serfozo, R.: Basics of applied stochastic processes. Springer (2009), doi:10.1007/978-3-540-89332-5
20. Skiena, S.: The Algorithm Design Manual (2. ed.). Springer (2008)
21. Torrini, P., Heckel, R., Ráth, I.: Stochastic simulation of graph transformation systems. In: Proc. Fundamental Approaches to Software Engineering (FASE 2010). LNCS, vol. 6013, pp. 154–157. Springer (2010)
22. Zhang, J., Chung, H.S., Hu, B.J.: Adaptive probabilities of crossover and mutation in genetic algorithms based on clustering technique. In: Proc. IEEE Congress on Evolutionary Computation (CEC 2004). pp. 2280–2287. IEEE (2004), doi:10.1109/CEC.2004.1331181