# Федеральное государственное автономное образовательное учреждение высшего профессионального образования «СИБИРСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Институт математики и фундаментальной информатики Базовая кафедра математического моделирования и процессов управления

## БАКАЛАВРСКАЯ РАБОТА

Направление 010101.62 Математика

## ПОРОЖДАЮЩЕЕ ВЕРОЯТНОСТНОЕ ПРОГРАММИРОВАНИЕ

Выпускник – Юрий Перов<sup>1</sup>.

Научный руководитель в СФУ – Татьяна Валерьевна Крупкина.

Кембридж-Оксфорд-Красноярск 2012–2014

<sup>&</sup>lt;sup>1</sup> Адрес для корреспонденции: <u>yuraperov@gmail.com</u>

#### РЕФЕРАТ

Выпускная квалификационная бакалаврская работа по теме «Порождающее вероятностное программирование» содержит 48 страниц текста, 50 использованных источников, 12 рисунков.

Ключевые слова: ВЕРОЯТНОСТНОЕ ПРОГРАММИРОВАНИЕ, МАШИННОЕ ОБУЧЕНИЕ, ИСКУССТВЕННЫЙ ИНТЕЛЛЕКТ, ВЕРОЯТНОСТНЫЕ МОДЕЛИ, АВТОМАТИЧЕСКОЕ МОДЕЛИРОВАНИЕ.

Работа посвящена новому направлению в области машинного обучения и компьютерных наук — вероятностному программированию. В работе дается краткое реферативное введение в языки вероятностного программирования Church/Venture/Anglican, а также описываются результаты первых экспериментов по автоматической генерации вероятностных программ.

# СОДЕРЖАНИЕ

B	ведеі	ние		4	
1	Краткое введение в языки вероятностного программирования Church,				
	Venture и Anglican				
	1.1 Первое знакомство с Church, Venture, Anglican			8	
	1.2	1.2 Статистический вывод в вероятностных языках программирования с п			
	щью алгоритма Метрополиса-Гастингса			11	
		1.2.1	Метод «выборки с отклонением»	11	
		1.2.2	Пространство историй выполнений вероятностных программ	12	
		1.2.3	Апостериорное распределение историй выполнений программ	14	
		1.2.4	Использование методов Монте-Карло по схеме Марковских цепей	15	
		1.2.5	Программная реализация статистического вывода	19	
	1.3	3 Эффективность вывода		21	
	1.4	4 Порождающее вероятностное программирование в распознавании образов		24	
	1.5	O различиях между Church, Venture, Anglican			
<b>2</b>	Авт	Автоматическая генерация вероятностных программ			
2.1 Обзор литературы		литературы	28		
	2.2	.2 Описание подхода		28	
	2.3	3 Грамматика и порождающие правила		33	
	2.4	4 Вероятности использования порождающих правил		35	
	2.5			35	
		2.5.1	Выборки из сгенерированных вероятностных программ	36	
		2.5.2	Вывод вероятностных программ, определяющих распределения, сов-		
			падающие или приближающие классические одномерные распреде-		
			пения	36	

	2.5.3	Обобщение произвольных эмпирических данных с помощью порож-			
		дающего вероятностного программирования	37		
	2.5.4	«Компиляция» вероятностных программ	39		
2.6	Обобі	цение порождающего вероятностного программирования	41		
Заключение					
Списо	Список использованных источников				

## **ВВЕДЕНИЕ**

Данная бакалаврская работа посвящена вероятностному программированию [1], новому направлению в областях машинного обучения, искусственного интеллекта и компьютерных наук, а именно реферативному краткому введению в вероятностное программирование, описанию языков вероятностного программирования Church [2], Venture [3] и Anglican [4], описанию подхода порождающего вероятностного программирования для решения задач распознавания образов [5], а также представлению полученных предварительных результатов по автоматизации вывода вероятностных моделей для вероятностного программирования [6].

#### Описание структуры работы

Первая часть данной работы начинается с общих сведений о вероятностном программировании, а затем в ней кратко описываются результаты, полученные коллегами автора и им самим в работе над научными проектами на протяжении двух лет в Массачусетском технологическом инситуте под руководством профессора Джошуа Тененбаума и доктора Викаша Мансингхи (Кембридж, штат Массачусетс, США) и Оксфордском университете под руководством профессора Френка Вуда (Великобритания, Оксфорд). Эта часть работы, являясь по сути переводом, представляет собой реферативную выдержку о вероятностном программировании и его приложениях на русском языке. Насколько автору известно, литературы о вероятностном программировании на русском языке практически нет, поэтому он надеется, что данная чисто реферативная часть работы принесет существенную пользу русскоговорящему научному сообществу, а особенно заинтересованным студентам и школьникам, которые впервые захотят познакомиться с развивающимся направлением вероятностного программирования.

Во второй части данной работы автор описывает новые результаты в области вероятностного программирования, связанные с автоматизированным или полуавтоматизированным выводом вероятностных моделей для вероятностного программирования, полученные во время стажировки автора в Департаменте технических наук Оксфордского

университета в научной лаборатории профессора Френка Вуда и под его руководством. Вторая часть завершается рассуждениями автора об обобщении автоматизированного изучения и вывода компьютером вероятностных моделей в виде вероятностных программ, то есть о возможностях порождающего вероятностного программирования.

#### Вероятностное программирование

Вероятностное программирование можно определить как компактный, композиционный способ представления порождающих вероятностных моделей и проведения статистического вывода в них с учетом данных с помощью обобщенных алгоритмов.

Вероятностная модель является важным понятием машинного обучения, одного из основных направлений искусственного интеллекта на сегодняшний день. В общем случае в рамках теории машинного обучения перед компьютером ставится задача произвести какое-то действие  $y_i$  на основе входных данных  $x_i$ , априорных знаний и возможности взаимодействовать со средой. Без ограничений общности целевым действием для компьютера можно считать производство ответа в виде выходных данных, представленных в виде информации. Например, в робототехнике эта информация может являться инструкциями для моторов и механических устройств для выполнения тех или иных физических действий роботом.

При использовании различных подходов в рамках машинного обучения используются модели, которые являются формальным «описанием компонентов и функций, отображающих существенные свойства моделируемого объема или процесса» [7]. В рамках машинного обучения используются вероятностные модели, так как свойства, элементы и связи между ними являются не фиксированными, а стохастическими.

Классическим примером одного из подходов в машинном обучении является способ «обучение с учителем», когда существует N прецедентов, то есть пар входных и выходных данных обучающей выборки  $X = \{x_i\}, Y = \{y_i\}, i = \overline{1,N},$  и необходимо найти алгоритм описывающий зависимость между  $x_i$  и  $y_i$ , то есть алгоритм F, который позволяет на основе каждого элемента входных данных  $x_i$  получать абсолютно или достаточно точный элемент выходных данных  $y_i$ , то есть  $F(x_i) \to y_i$ . С помощью данного алгоритма затем для M известных элементов входных данных  $\{x_i\}, j = \overline{N+1,N+M}$  находятся значения M неизвестных элементов выходных данных  $\{y_i\}, j = \overline{N+1,N+M}$ . В рамках машинного обучения эта проблема, в том числе, решается с помощью методов регрессионного

анализа.

В только что приведенной как пример модели неизвестными (скрытыми) параметрами  $T=\{t_j\}, j=\overline{1,K}$  будет информация о характеристиках модели F, которые подлежат выводу при данной обучающей выборке в виде пар  $(x_i,y_i)$ .

Приведем простой пример: в линейной регрессии (при L независимых переменных) значениями независимых переменных будут  $x_{i,j}$ , значениями зависимой переменной будут  $y_i$ , параметрами модели будут  $t_1, t_2, \ldots, t_{L+1}$ , а алгоритмом F будет

$$y_i = F_{t_1,\dots,t_{L+1}}(x_i) = t_1 + t_2 \cdot x_{i,1} + \dots + t_{L+1} \cdot x_{i,L}.$$

Для простоты без ограничений общности будем считать, что мы можем всегда получить  $y_i$  детерминированно, зная T и  $x_i$ .

В порождающих вероятностных моделях задается совместное распределение вероятностей P(T,X), обычно сначала путем задания априорного распределения P(T), а затем задания условного распределения  $P(X\mid T)$ . Это и называется моделью.

При заданной модели и известных X задачей будет являться поиск апостериорного распределения на T, таким образом  $P(T\mid X)$ . Одним из способов поиска данного апостериорного распределения является применение теоремы Байеса:

$$P(T \mid X) = \frac{P(T)P(X \mid T)}{P(X)},$$

где P(X) теоретически можно найти как  $\int P(T)P(X\mid T)\;dT$ , но при решении практических задач это часто невозможно, так как перебор всего пространства T не поддается аналитическому решению или решению с помощью численных методов за разумное время. Поэтому чаще всего при решении задач в рамках Байесовского подхода стоит задача поиска ненормированного значения  $P(T\mid X)$ :

$$P(T \mid X) \propto P(T)P(X \mid T),$$

где символ  $\propto$ , часто встречающийся в зарубежной литературе, но очень редко встречающийся у нас, означает «пропорционально». Нормировочную константу затем можно найти приближенно, но иногда ее значение даже не вычисляют, так как бывает достаточно найти и работать дальше с наиболее вероятными элементами  $\hat{T}$  из апостериорного распределения

 $P(T \mid X)$ .

Обычно, особенно при решении практических задач с большим объемом данных и в рамках сложных моделей, апостериорное распределение  $P(T \mid X)$  находят не точно, а с помощью приближенных методов, в том числе с помощью методов Монте-Карло [8], которые позволяют сгенерировать выборку из интересующего нас распределения.

Как отмечалось в самом начале данного подраздела, вероятностное программирование позволяет:

- 1. Композиционно и компактно записывать порождающую вероятностную модель с помощью задания априорных вероятностей P(T) и  $P(X \mid T)$  в виде алгоритма (вероятностной программы) с использованием стохастических функций (например, стохастическая функция  $Normal(\mu, \sigma)$ ) и вспомогательных детерминированных функций (например, + или \*).
- 2. Снабжать модель данными  $\hat{X}$ , таким образом теоретически определяя условное распределение  $P(T\mid \hat{X})$ .
- 3. Производить статистический вывод для генерации выборки из условного распределения с помощью обобщенных алгоритмов статистического вывода, подходящих для всех или большого множества моделей.

Для ознакомления с машинным обучением и искусственным интеллектом автор рекомендует следующие источники: [9, 10, 11]. Информацию о вероятностных моделях и Байесовских методах на русском языке можно найти в [12].

# 1 Краткое введение в языки вероятностного программирования Church, Venture и Anglican

Существует более 15 языков вероятностного программирования, перечень с кратким описанием каждого из них можно найти на [13]. В данной работе реферативно будут рассмотрены три языка вероятностного программирования: Church [2], Venture [3] и Anglican [4]. Языки Venture и Anglican являются продолжениями языка Church. Church в свою очередь основан на языке «обычного» программирования Lisp и Scheme. Заинтересованному читателю крайне рекомендуется ознакомиться с книгой [14], являющейся одним из лучших способов начать знакомство с языком «обычного» программирования Scheme.

#### 1.1 Первое знакомство с Church, Venture, Anglican

На текущий момент любой язык вероятностного программирования напрямую связан с методами статистического вывода, которые используются в нем, поэтому часто язык ассоциируется с платформой вероятностного программирования, то есть с его технической реализацией в виде компьютерной программы.

Рассмотрим задание простой вероятностной модели Байесовской линейной регрессии [10] на языке вероятностного программирования Venture/Anglican [15] в виде вероятностной программы:

```
1 [ASSUME t1 (normal 0 1)]
2 [ASSUME t2 (normal 0 1)]
3 [ASSUME noise 0.01]
4 [ASSUME noisy_x (lambda (time) (normal (+ t1 (* t2 time)) noise))]
5 [OBSERVE (noisy_x 1.0) 10.3]
6 [OBSERVE (noisy_x 2.0) 11.1]
7 [OBSERVE (noisy_x 3.0) 11.9]
8 [PREDICT t1]
9 [PREDICT t2]
0 [PREDICT (noisy_x 4.0)]
```

Скрытые искомые параметры — значения коэффициентов  $t_1$  и  $t_2$  линейной функции  $x\left(time\right) = t_1 + t_2 \cdot time$ . У нас есть априорные предположения о данных коэффициентах,

а именно мы предполагаем, что они распределены по закону нормального распределения  $\operatorname{Normal}(0,1)$  со средним 0 и стандартным отклонением 1. Таким образом, мы определили в первых двух строках вероятностной программы вероятность P(T), описанную в предыдущем раздел. Инструкцию [ASSUME name expression] можно рассматривать как определение случайной величины с именем name, принимающей значение вычисляемого выражение (программного кода) expression, которое содержит в себе неопределенность.

Вероятностные языки программирования (здесь и далее будут иметься в виду конкретно Church, Venture, Anglican, если не указано иное), как и Lisp/Scheme, являются функциональными языками программирования, и используют польскую нотацию<sup>1</sup> при записи выражений для вычисления. Это означает, что в выражении вызова функции сначала располагается оператор, а уже только потом аргументы: (+ 1 2), и вызов функции обрамляется круглыми скобками. На других языках программирования, таких как C++ или Python, это будет эквивалентно коду 1 + 2.

В вероятностных языках программирования выражение вызова функции принято разделять на три разных вида:

- 1. Вызов детерминированных процедур (primitive-procedure arg1 ...argN), которые при одних и тех же аргументах всегда возвращают одно и то же значение. К таким процедурам, например, относятся арифметические операции.
- 2. Вызов вероятностных (стохастических) процедур (stochastic-procedure arg1 ...argN), которые при каждом вызове генерируют случайным образом элемент из соответствующего распределения. Такой вызов определяет новую случайную величину. Например, вызов вероятностной процедуры (normal 1 10) определяет случайную величину, распределенную по закону нормального распределения Normal $(1, \sqrt{10})$ , и результатом выполнения каждый раз будет какое-то вещественное число.
- 3. Вызов составных процедур (compound-procedure arg1 ...argN), где compound-procedure введенная пользователем процедура с помощью специального выражения lambda: (lambda (arg1 ...argN) body), где body тело процедуры, состоящее из выражений. В общем случае составная процедура является стохастической (недетерминированной) составной процедурой, так как

 $<sup>^{1}</sup>$ Venture имеет отдельный дополнительный вид синтаксиса VentureScript, использующий инфиксную нотацию и не требующий обрамления вызова функций скобками, то есть схожий по своей сути с привычными большинству людей языками программирования C, C++, Python и т.д.

ее тело может содержать вызовы вероятностных процедур.

После этого мы хотим задать условную вероятность  $P(X \mid T)$  наблюдаемых переменных  $x_1, x_2, x_3$  при заданных значениях скрытых переменных  $t_1, t_2$  и параметра time.

Перед вводом непосредственно самих наблюдений с помощью выражения [OBSERVE . . . .] мы определяем общий закон для наблюдаемых переменных  $\{x_i\}$  в рамках нашей модели, а именно мы предполагаем, что данные наблюдаемые случайные величины при заданных  $t_1, t_2$  и заданном уровне шума noise распределены по закону нормального распределения Normal $(t_1+t_2\cdot time, \sqrt{noise})$  со средним  $t_1+t_2\cdot time$  и стандартным отклонением noise. Данная условная вероятность определена на строках 3 и 4 данной вероятностной программы.  $noisy_x$  определена как функция, принимающая параметр time и возвращающая случайное значение, определенное с помощью вычисления выражение (normal (+ t1 (\* t2 time)) noise) и обусловленное значениями случайных величин  $t_1$  и  $t_2$  и переменной noise. Отметим, что выражение (normal (+ t1 (\* t2 time)) noise) содержит в себе неопределенность, поэтому каждый раз при его вычислении мы будем получать в общем случае разное значение.

На строках 5—7 мы непосредственно вводим известные значения  $\hat{x_1}=10.3, \, \hat{x_2}=11.1, \, \hat{x_3}=11.9.$  Инструкция вида [OBSERVE expression value] фиксирует наблюдение о том, что случайная величина, принимающая значение согласно выполнению выражения expression, приняла значение value.

Повторим на данном этапе всё, что мы сделали. На строках 1-4 с помощью инструкций вида [ASSUME . . . ] мы задали непосредственно саму вероятностную модель: P(T) и  $P(X \mid T)$ . На строках 5-7 мы непосредственно задали известные нам значения наблюдаемых случайных величин X с помощью инструкций вида [OBSERVE . . . ].

На строках 8—9 мы запрашиваем у системы вероятностного программирования апостериорное распределение  $P(T \mid X)$  скрытых случайных величин  $t_1$  и  $t_2$ . Как уже было сказано, при большом объеме данных и достаточно сложных моделях получить точное аналитическое представление невозможно, поэтому инструкции вида [PREDICT . . . ] генерируют выборку значений случайных величин из апостериорного распределения  $P(T \mid X)$  или его приближения. Инструкция вида [PREDICT expression] в общем случае генерирует один элемент выборки из значений случайной величины, принимающей значение

согласно выполнению выражения **expression**. Если перед инструкциями вида [PREDICT ...] расположены инструкции вида [OBSERVE ...], то выборка будет из апостериорного распределения<sup>2</sup>, обусловленного перечисленными ранее введенными наблюдениями.

Отметим, что в завершении мы можем также предсказать значение функции x(time) в другой точке, например, при time = 4.0. Под предсказанием в данном случае понимается генерация выборки из апостериорного распределения новой случайной величины при значениях скрытых случайных величин  $t_1, t_2$  и параметре time = 4.0.

Для генерации выборки из апостериорного распределения  $P(T \mid X)$  в языке программирования Church в качестве основного используется алгоритм Метрополиса-Гастингса, который относится к методам Монте-Карло по схеме Марковских цепей. В следующем подразделе будет произведено подробное описание применения данного алгоритма для обобщенного статистического вывода в вероятностных языках. Под «обобщенным» выводом в данном случае понимается то, что алгоритм может быть применен к любым вероятностным программам, написанным на данном вероятностном языке программирования.

# 1.2 Статистический вывод в вероятностных языках программирования с помощью алгоритма Метрополиса-Гастингса

Описание алгоритма Метрополиса-Гастингса в применении к «семейству» вероятностных языков Church впервые опубликовано в [2] и более подробно описано в [16].

Получить выборку из N элементов из априорного распределения скрытых параметров P(T), наблюдаемых величин  $P(X \mid T)$  или их совместного распределение P(T,X) какой-либо порождающей вероятностной модели, записанной в виде вероятностной программы, не составляет труда. Для этого достаточно выполнить вероятностную программу N раз. Отметим очевидный факт, что в данном случае вероятностная программа будет содержать лишь инструкции вида [ASSUME . . . ] (задание вероятностной модели) и [PREDICT . . . . ] (перечисление случайных величин, выборку которых мы генерируем).

#### 1.2.1 Метод «выборки с отклонением»

При заданных наблюдениях с помощью инструкций вида [OBSERVE expression value] наиболее простым способом получения апостериорного распределения является метод

<sup>&</sup>lt;sup>2</sup>Говоря точнее, конечно, из приближения апостериорного распределения.

«выборки с отклонением» [17]. Для понимания рассмотрим следующую вероятностную программу:

```
1 [ASSUME a (uniform-discrete 1 6)]
2 [ASSUME b (uniform-discrete 1 6)]
3 [OBSERVE (+ a b) 5]
4 [PREDICT (* a b)]
```

Отметим, что стохастическая процедура (uniform-continuous a b) возвращает значение случайной величины, распределенной по равномерному дискретному закону распределения с носителем  $\{a, \ldots, b\}$ .

Текстом задачу, записанную выше с помощью вероятностной программы, можно сформулировать следующим образом: подбрасываются два шестигранных игральных кубика, в сумме выпало 5; каково распределение произведения очков на этих двух кубиках?

Метод «выборки с отклонением» заключается в том, чтобы генерировать значения очков на первом и втором кубиках из их априорного распределения (таким образом, два независимых дискретных равномерных распределения) и проверять, равна ли их сумма 5. Если нет, данная попытка отвергается и не учитывается. Если да, то произведение очков на кубиках добавляется во множество элементов выборки. Данная выборка и будет аппроксимацией апостериорного распределения значений произведения очков на двух кубиках, если известно, что сумма очков равна 5.

Данный метод является неэффективным, и при решении более сложных задач просто вычислительно неосуществимым. Также отметим, что он хорошо подходит только для дискретных значений случайных и промежуточных переменных.

#### 1.2.2 Пространство историй выполнений вероятностных программ

При выполнении вероятностной программы каждая случайная величина принимает определенное значение. Например, следующая вероятностная программа содержит три случайных величины, каждая из которых распределена по закону нормального распределения:

```
1 [ASSUME a (normal 0 1)]
2 [ASSUME b (normal 0 1)]
```

```
3 [ASSUME c (normal (+ a b) 1)]
4 [PREDICT c]
```

При выполнении данной вероятностной программы каждая из трех случайных величин примет свое случайное значение. Назовем *историей выполнения* вероятностной программы отображение, сопоставляющее каждой случайной величине ее значение. Мощность *истории выполнения* совпадает с количеством случайных величин, которые были созданы при ее выполнении.

Для каждой программы можно определить соответствующее вероятностное пространство, элементарными событиями которого будут являться все *истории ее выполнения*. Вероятностью каждого элементарного события будет являться произведение вероятностей принятие того или иного значения каждой случайные величины при данной истории выполнения.

Очевидно, что история выполнения однозначным образом определяет выполнение вероятностной программы и принимаемые значения, как случайных, так и зависящих от них детерминированных выражений.

Для примера рассмотрим еще более простую вероятностную программу.

```
1 [ASSUME a (bernoulli 0.7)]
2 [ASSUME b (bernoulli 0.7)]
3 [PREDICT a]
4 [PREDICT b]
```

В ней обе случайных величины имеют свое название, что облегчает запись историй выполнений (иначе необходимо вводить адресную схему для наименования случайных величин, чтобы однозначно идентифицировать их).

Данная вероятностная программа имеет четыре различных возможных историй выполнений, а именно  $\{a:0,b:0\}, \{a:0,b:1\}, \{a:1,b:0\}$  и  $\{a:1,b:1\}$  с вероятностями 0.09, 0.21, 0.21 и 0.49 соответственно.

Отметим также, что количество «активных» случайных величин при выполнении одной и той же вероятностной программы может меняться, как, например, в следующей программе:

```
[ASSUME geometric
```

Данная программа генерирует элемент (т.е. одноэлементную выборку) из геометрического распределения с параметром 0.5 с помощью составной процедуры geometric, параметризованной параметром p. При выполнении данной вероятностной программы в истории ее выполнения количество «активных» случайных величин может быть любым.

#### 1.2.3 Апостериорное распределение историй выполнений программ

При добавлении наблюдений, то есть фиксации значения определенных случайных величин, можно считать, что рассматривается подмножество множества элементарных событий, а именно те элементы, в историях выполнений которых наблюдаемые случайные величины принимают желаемое значение.

Можно описать другое вероятностное пространство, множеством элементарных событий которого будет являться только что описанное подмножество. Вероятностная мера в новом вероятностном пространстве может быть «индуцирована» вероятностной мерой из первоначального вероятностного пространства с учетом нормировочной константы.

Только что описанный переход полностью сочетается с теоремой Байеса, которая в нашем случае записывается следующим образом:

$$P(T \mid X = \hat{x}) = \frac{P(T)P(X = \hat{x} \mid T)}{P(X = \hat{x})},$$

где X — множество случайных величин, для которых мы знаем фиксированные наблюдаемые значения; T — множество случайных величин, ассоциируемых со скрытыми параметрами, апостериорное распределение которых мы заинтересованы получить (грубо говоря, это всё остальные случайные величины);  $\hat{x}$  — наблюдаемые значения случайных величин.

Отметим, что часто мы заинтересованы не в апостериорном распределении  $P(T \mid X = \hat{x})$ , а в апостериорном распределении лишь части скрытых параметров  $P(T' \subset T \mid X = \hat{x})$  или даже функции f от них  $P(f(T) \mid X = \hat{x})$ , в общем случае не являющейся биекцией. С другой стороны, так как мы не ставим задачу получить аналитическое представление данных апостериорных распределений, а лишь выборку из них, то

это не играет большой роли в нашем случае при решении задач методами Монте-Карло: мы можем генерировать элементы выборки из  $P(T\mid X=\hat{x})$ , и затем использовать только значения нужных нам скрытых случайных величин и/или действовать функцией f на них.

#### 1.2.4 Использование методов Монте-Карло по схеме Марковских цепей

Математическо-статистический аппарат методов Монте-Карло по схеме Марковских цепей кратко и «современно» изложен в [8].

Интуитивно опишем, что мы собираемся делать. У нас есть вероятностная программа g, определяющая множество скрытых T и наблюдаемых X случайных величин. Вероятностная программа своей записью задает априорное распределение  $P(T), P(X \mid T)$ , а значит и совместное распределение P(T,X). Каждый раз путем выполнения данной программы мы получаем одну из возможных реализаций данной программы, которая биективно описывается соответствующей историей выполнения  $h_i \in H$ , где множество H — множество всех возможных историй выполнений данной вероятностной программы, которое можно рассматривать как множество элементарных событий. Для каждой истории выполнения определена ее вероятностная мера

$$P(h_i) = P(T = \tilde{t}, X = \tilde{x}) = P(T = \tilde{t})P(X = \tilde{x} \mid T = \tilde{t}).$$

У нас также есть «экспериментальное» значение каждой наблюдаемой случайной величины  $\hat{x}_i$ , то есть множество  $\{\hat{x}_i\} = \hat{x}$ . Существует подмножество историй выполнений  $H' \subset H$ , в котором наблюдаемые случайные величины принимают желаемое значение. Данное подмножество можно рассматривать как множество элементарных событий другого вероятностного пространства, вероятностную меру на котором можно «индуцировать» из предыдущего путем деления на нормирующую постоянную  $P(X = \hat{x})$ :

$$P(h') = \frac{P(T = \tilde{t})P(X = \hat{x} \mid T = \tilde{t})}{P(X = \hat{x})}.$$

Будем обозначать ненормированную вероятностную меру

$$P(\tilde{h}') = P(T = \tilde{t})P(X = \hat{x} \mid T = \tilde{t}).$$

Рассмотрим цепь Маркова, исходами которой являются истории выполнений  $h' \in H'$ . В качестве начального распределения можно выбрать априорное распределение  $P(T)P(X\mid T)$ , при котором значения наблюдаемых случайных величин не выбираются случайным образом согласно их распределению, а устанавливаются согласно их значениям. Например, в вероятностной программе

- 1 [ASSUME a (gamma 1 1)]
- 2 [ASSUME b (lambda () (normal a 1))]
- 3 [OBSERVE (b) 5.3]

первая случайная величина a, распределенная по закону Гамма-распределения Gamma(1,1), будет являться скрытой, и будет сгенерирована согласно данному закону распределения. Наблюдаемая же случайная величина, распределенная по нормальному закону Normal(a,1), не будет сгенерирована, а будет установлена в соответствии c ее наблюдаемым значением.

Мы хотим установить такие правила перехода по схеме Метрополиса-Гастингса из одного состояния (исхода) цепи Маркова в другое, чтобы стационарное распределение данной цепи Маркова совпадало с распределением P(h'). В таком случае для получения аппроксимации искомого апостериорного распределения в виде выборки нам будет достаточно имитировать данную цепь Маркова [8, 18].

В алгоритме Метрополиса-Гастингса вероятностная мера может быть известна с точностью до нормировочной константы, что и происходит в нашем случае. На каждом шаге алгоритма дано текущее состояние  $h'_t$  и в соответствии с заданным заранее условным распределение предлагается новое состояние  $h'^* \sim Q(\;\cdot\;|\;h'_t)$ . Таким образом, Q можно назвать распределением предлагаемых переходов. После этого подсчитывается коэффициент «принятия» нового состояния:

$$\alpha = \min \left( 1, \frac{\tilde{P}(h'^*) \ Q(h'_t \mid h'^*)}{\tilde{P}(h'_t) \ Q(h'^* \mid h'_t)} \right).$$

Состояние  $h'^*$  принимается в качестве следующего состояния  $h'_{t+1}$  с вероятностью  $\alpha$ , в противном случае  $h'_{t+1}:=h'_t$ .

Новое состояние предлагается следующим образом:

1. Случайным образом (равномерно) выбирается одна «активная» скрытая случайная величина  $r \in \mathrm{dom}\, h_t'$  для «вариации».

- 2. Предлагается новое значение данной случайной величины  $r^* \sim \kappa( \cdot \mid r = \hat{r}, h'_t)$ . Условная вероятность  $\kappa$  в данном случае будет локальным для r распределением предлагаемых переходов.
- 3. Если случайная величина r влияет на поток выполнения вероятностной программы, и в результате ее нового значения должны быть исполнены другие ветви выполнения вероятностной программы, они исполняются и в общем случае происходит генерация новых случайных величин, которые прежде были неактивны.

Для примера рассмотрим следующую вероятностную программу:

```
1 [ASSUME a (bernoulli 0.3)]
```

- 2 [ASSUME b (if (= a 1) (normal 0 1) (gamma 1 1))]
- B [PREDICT c]

В данной вероятностной программе три случайных величины, первая  $\xi_1$  распределена по закону Бернулли, вторая  $\xi_2$  по закону нормального распределения, третья  $\xi_3$  по закону Гамма-распределения.  $\xi_1$  можно также называть «а» (по имени переменной), хотя переменная «b» не является сама по себе случайной величиной, а зависит от значения  $\xi_1$ , определяющей поток выполнения вероятностной программы, и от значения либо  $\xi_2$ , либо  $\xi_3$ . Отметим также, что при каждом выполнении данной вероятностной программы будет существовать только две активных случайных величины.

Предположим, что текущим состоянием вероятностной программы в момент времени t была история выполнения  $h'_t = \{\xi_1 = 1, \xi_2 = 3.2\}$ . Это означает, что случайная величина  $\xi$  приняла значение 1, и поэтому для задания переменной b был сгенерирован элемент из нормального распределения, то есть была «реализована» случайная величина  $\xi_2 \sim \text{Normal}(0,1)$ . Очевидно, что

$$P(h_t') = P(\xi_1 = 1) \cdot P(\xi_2 = 3.2 \mid \xi_1 = 1) = 0.3 \cdot f_{\text{Normal}(0,1)}(3.2) \approx 0.0007152264603.$$

Предложим новое состояние  $h'^*$ :

- 1. Выберем случайным образом одну из двух «активных» случайных величин, пусть это будет  $\xi_1$ .
- 2. Предложим случайным образом новое значение данной случайной величины согласно ее априорному распределению (то есть Bernoulli(0.3)), пусть это будет 0.
- 3. Так как случайная величина  $\xi_1$  действительно влияет на поток выполнения ве-

роятностной программы и при изменении ее значения в данном случае должна быть выполнена другая ветвь, выполним данную ветвь, генерируя значения «активирующихся» случайных величин (в нашем случае только  $\xi_3$ ). Предположим, что  $\xi_3$  стало равным 12.3.

Тогда

$$Q(h'^* \mid h'_t) = \frac{1}{2} \cdot 0.7 \cdot f_{\text{Gamma}(1,1)}(12.3).$$

В общем случае

$$Q(h'^* \mid h'_t) = \frac{1}{|h'_t|} \cdot \kappa(r^* \mid r = \hat{r}, h'_t) \cdot R_{new},$$

где  $|h_t'|$  — количество активных случайных величин в текущей истории выполнения,  $\kappa(r^* \mid r = \hat{r}, h_t')$  — вероятность нового значения варьируемой случайной величины  $r, R_{new}$  — совместная вероятность выбора своих значений «активирующихся» случайных величин.

При уже фиксированном  $h'^*$  обратное  $Q(h'_t \mid h'^*)$  для коэффициента принятия алгоритма Метрополиса-Гастингса в общем случае может быть посчитано аналогичным образом:

$$Q(h'_t \mid h'^*) = \frac{1}{|h'^*|} \cdot \kappa(\hat{r} \mid r = r^*, h'^*) \cdot R_{old},$$

где  $R_{old}$  — совместная вероятность выбора своих значений случайных величин, активных в  $h_t'$ , но не являющихся активными в  $h'^*$ .

В только что рассмотренном примере

$$Q(h'_t \mid h'^*) = \frac{1}{2} \cdot 0.3 \cdot f_{\text{Normal}(0,1)}(3.2).$$

И тогда с учетом того, что  $P(h'^*)=0.7\cdot f_{\mathrm{Gamma}(1,1)}(12.3)\approx 0.000003186221124,$  мы получаем, что

$$\alpha = \min\left(1, \frac{0.7 \cdot f_{\text{Gamma}(1,1)}(12.3) \cdot \frac{1}{2} \cdot 0.3 \cdot f_{\text{Normal}(0,1)}(3.2)}{0.3 \cdot f_{\text{Normal}(0,1)}(3.2) \cdot \frac{1}{2} \cdot 0.7 \cdot f_{\text{Gamma}(1,1)}(12.3)}\right) = 1.$$

Поэтому в данном конкретном примере  $h'_{t+1} = h'^*$  в любом случае, то есть с вероятностью один.

Выбор локального для r распределения предлагаемых переходов может быть раз-

ным. В простейшем случае, если r — независимая случайная величина, данное распределение выбирается идентичным априорному распределению для r. Если же r — перестановочная случайная величина [19], а такие случайные величины поддерживаются рассматриваемыми языками вероятностного программирования, то ее распределение может быть выбрано с учетом уже накопленных значений.

Таким образом, мы описали алгоритм для предложения нового состояния  $h'^*$  при текущем состоянии  $h'_t$  и описали его в рамках метода Метрополиса-Гастингса. Чтобы получить выборку  $\{h'_j\}_{j=1}^N$  из N элементов желаемого апостериорного распределения, нам необходимо имитировать данную цепь Маркова согласно описанному выше алгоритму и получить  $\{h'_j\}_{l=j}^{M+N\cdot K}$  элементов данной цепи. Затем необходимо отсеять первые M элементов и из оставшихся выбрать каждый N-й элемент. Полученное множество будет являться аппроксимацией искомой выборки  $\{h'_j\}_{j=1}^N$  [8], а так как любая история выполнений h' однозначно определяет значение всех случайных величин в вероятностной программе, то и аппроксимацией выборки из  $P(T \mid X = \hat{x})$ .

K выбирается больше единицы, чтобы исключить автокорреляцию  $h_i'$ , которая естественным образом возникает в цепи Маркова. M выбирается достаточно большим, чтобы независимо от выбора начальной точки  $h_1'$  цепь успела «забыть» данный первоначальный выбор; это особенно важно, когда начальное априорное распределение P(T,X) очень сильно отличается от апостериорного  $P(T\mid X=\hat{x})$ . Какого-то общего правила для выбора данных величин нет, и обычно они выбираются эмпирически.

#### 1.2.5 Программная реализация статистического вывода

В предыдущем подпункте теоретически был описан алгоритм для обобщенного статистического вывода в вероятностных языках программирования.

Простая программная реализация впервые достаточно подробно была описана в [16]:

1. Для инициализации  $h'_1$  выбирается случайным образом путем выполнения вероятностной программы и фиксации наблюдаемых случайных величин в соответствии с имеющимися данными. При этом в памяти компьютера сохраняется база данных активных случайных величин вместе с их значениями, а также сохраняется  $P(h'_1)$ , которое подсчитывается во время первоначального выполнения программы с учетом вероятности наблюдаемых случайных величин принять то

- значение, которое они приняли. В [16] описывается схема адресации, которая позволяет различать случайные величины между собой.
- 2. Затем для выбора последующего  $h'_{t+1}$  каждый раз из базы данных случайным равномерным образом выбирается случайная величина r и она случайным образом варьируется в соответствии со своим локальным распределением предлагаемых переходов  $\kappa(\cdot \mid r = \hat{r}, h'_t)$ , которое предварительно задается для всех используемых примитивных (несоставных) случайных величин. Создается копия базы данных, в которой значение случайной величины r заменяется на новое.
- 3. Вероятностная программа выполняется еще один раз, при этом случайные величины, для которых уже в базе данных имеется запись (согласно схеме адресации), принимают соответствующие старые значения, а r принимает свое новое полученное значение.
- 4. В общем случае, если r влияет на поток выполнения программы, генерируются значения для новых случайных величин, которые становятся активными. Для этих случайных величин прежде в базе данных не было записей.
- 5. Также если r влияет на поток выполнения программы, то некоторые случайные величины могут перестать быть активными. В данном случае соответствующие им записи в базе данных будут невостребованы.
- 6. После выполнения программы второй раз можно считать, что мы получили  $h'^*$ . Также отметим, что при выполнении программы  $h'^*$  мы получили и использовали всё необходимые компоненты для подсчета  $P(h'^*)$  и  $Q(h'^* \mid h'_t)$ .
- 7. Вероятность обратного перехода  $Q(h'_t \mid h'^*)$  может быть получена разными способами. Если в вероятностной программе есть только независимые случайные величины (и нет перестановочных), то первые две компоненты  $Q(h'_t \mid h'^*)$  могут быть найдены тривиально, так как мы знаем количество случайных величин в  $h'^*$  и можем посчитать  $\kappa(\hat{r} \mid r = r^*, h'^*)$ . Для третьей компоненты нам нужно посчитать произведение вероятностей случайных величин, которые перестали быть активными в  $h'^*$ , то есть все невостребованные случайные величины в базе данных. В случае наличия и использования перестановочных случайных величин мы можем имитировать ситуацию, что  $h'^*$  является нашим старым состоянием, а новое состояние мы получаем в точном соответствии с  $h'_t$  [3].

- 8. Имея  $P(h'_t)$ ,  $P(h'^*)$ ,  $Q(h'^* \mid h'_t)$  и  $Q(h'_t \mid h'^*)$ , мы подсчитываем коэффициент принятия  $\alpha$  для алгоритма Метрополиса-Гастингса, и либо принимаем  $h'^*$  с вероятностью  $\alpha$ , что означает, что в следующий раз мы будем использовать уже новую базу данных случайных величин со значением, соответствующими  $h'^*$ , либо отклоняем с вероятностью  $(1-\alpha)$ , что означает, что на следующем шаге  $h'_{t+1}$  мы снова будем использовать старую базы данную от  $h'_t$ .
- 9. Алгоритм повторяется с шага № 2.

#### 1.3 Эффективность вывода

При использовании вероятностных языков программирования встает вопрос об эффективности статистического вывода, другими словами о том, как быстро мы генерируем выборку желаемой точности из апостериорного распределения. Описанная в секции 1.2.5 программная реализация статистического вывода является неэффективной, так как происходит перевыполнение всей вероятностной программы, хотя вариация случайной величины r обычно имеет только локальный эффект.

Рассмотрим более подробно данную проблему на примере следующей вероятностной программы:

```
1
   [ASSUME rainy-season (bernoulli 0.2)]
   [ASSUME cloudy
2
3
     (bernoulli (if rainy-season 0.8 0.3))]
   [ASSUME rain
4
     (bernoulli (if cloudy 0.8 \ 0.2))]
5
6
   [ASSUME sprinkler
     (bernoulli (if cloudy 0.1 0.5))]
7
   [ASSUME wet-grass
8
9
     (bernoulli
       (if sprinkler (if rain 0.99 0.9)
10
11
                       (if rain 0.9 0.01)))]
```

Данная вероятностная программа описывает статистически простую упрощенную модель зависимости между сезоном, облачностью, дождем, работой разбрызгивателя и состоянием травы (мокрая или нет) в какой-то день. Значение переменных следующее:

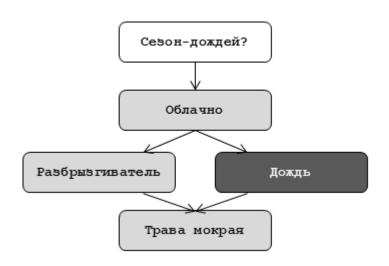


Рисунок 1 — Пример Байесовской сети. Из [20].

rainy-season: входит ли тот день в сезон дождей или нет?, cloudy: облачно в тот день или нет?; rain: был ли дождь в тот день?; sprinkler: работал ли разбрызгиватель в тот день или нет?; wet-grass: была ли трава мокрой в тот день? Данная модель может быть представлена с помощью Байесовской сети доверия (см. рис. 1) и таблицами условных вероятностей. Отметим, что любая Байесовская сеть доверия может быть представлена в виде вероятностной программы на языке Church/Venture/Anglican, но не любая вероятностная программа может быть представлена Байесовской сетью.

В случае, если во время очередной итерации алгоритма Метрополиса-Гастингса в качестве варьируемой случайной величины выбрана случайная величина rain (соответствующий узел на рис. 1 выделен самым темным цветом), для подсчета коэффициента принятия достаточно лишь рассмотреть значения и вероятности при данных значениях узлов «Дождь» и «Трава мокрая». Все остальные значения и их вероятности останутся прежними. Иллюстрацию распространения возмущений в связи с вариацией случайной величины «Дождь» см. на рис. 2.

В подобном простом примере этот факт не играет большой роли, так как отношение случайных величин, требующих «переучета», к общему количеству активных случайных величин невелико, однако при большом объеме данных это играет решающую роль при выполнении статистического вывода во многих моделях при помощи программных реализаций вероятностных языков программирования. Например, в скрытых марковских моделях [17, 2, 4] или в латентном размещении Дирихле [21, 20].

Например, в простой скрытой Марковской модели есть N скрытых и N наблюдаемых величин. При реализации статистического вывода алгоритмом, описанным в сек-

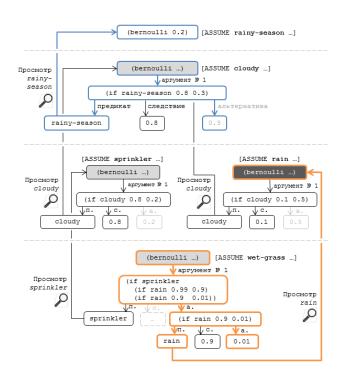


Рисунок 2 — Байесовская сеть в виде «отпечатка» выполнения вероятностной программы. Из [20].

ции 1.2.5, одна итерация алгоритма Метрополиса-Гастингса имеет сложность (по времени) O(N), хотя желаемая и возможная сложность O(1) или по крайней мере  $O(\log N)$ .

Для достижения желаемой сложности в каждой истории выполнений необходимо отслеживать зависимости между случайными величинами. Следует отметить, что эти зависимости в общем случае могут изменяться в вероятностных программах. Например, в следующей программе

- 1 [ASSUME a (bernoulli 0.5)]
- $2 \quad [ASSUME b \quad (gamma 1 \quad 1)]$
- B [ASSUME c (normal (if a b 3.0) 1)]

случайная величина c, т.о. (normal ...), зависит от случайной величины b, т.о. (gamma ...), не всегда, а только если случайная величина a принимает значение «ИСТИ-НА».

Описание структур данных и алгоритмов, необходимых для отслеживания зависимостей в режиме реального времени, были предварительно приведено в [20] и [22], а затем более обширно и подробно в [3]. При использовании данных структур данных и алгоритмов временная сложность одной итерации алгоритма Метрополиса-Гастингса в простой скрытой Марковской модели равна  $O(\log N)$ , при этом если использовать упорядоченный

перебор случайных величин, то временная сложность снизиться до O(1), так как логарифмический фактор появляется в связи с необходимостью выбирать случайным образом следующий узел (т.е. случайную величину) для вариации.

В простой скрытой Марковской модели количество «активных» случайных величин постоянно и вид зависимости тривиален (от каждой скрытой случайной величины напрямую зависит только одна наблюдаемая случайная величина и только следующая скрытая случайная величина). В более сложных моделях виды зависимостей более изощрены и количество случайных величин меняется от одной истории выполнений к другой. С другой стороны, для большого количества порождающих моделей, используемых в настоящее время в машинном обучении, необходимо, чтобы время на одну итерацию оставалось постоянным или росло хотя бы логарифмически вместе с линейным ростом количества наблюдений, иначе статистический вывод будет невозможен за разумное время.

На рис. З показаны результаты применения алгоритмов и структур данных, описанных в [20, 22, 3]. При использовании старого подхода, описанного в [16] (см. секцию 1.2.5) время на N итераций алгоритма Метрополиса-Гастингса росло квадратично с линейным ростом размерности модели (N скрытых и N наблюдаемых случайных величин), а при использовании предлагаемого подхода время растет квазилинейно.

На этом же рисунке видно, что благодаря локализации выполнения одной итерации Метрополиса-Гастингса стало возможным проводить приближенный статистический вывод параллельно.

Следует отметить, что пространственная сложность алгоритма (объем памяти, необходимый для его применения) равна, грубо говоря, O(N+K), где O(K) — стоимость хранения в памяти зависимостей между случайными величинами. В общем случае эта величина может быть достаточно большой, и в работе [22] описаны возможности более эффективного расходования памяти.

## 1.4 Порождающее вероятностное программирование в распознавании образов

Как было отмечено в начале, цель вероятностного программирования — облегчить задачу моделирования порождающих вероятностных моделей и проведения вывода в них. Примером иллюстрации успешного предварительного применения вероятностного программирования может служить работа [5], в которой вероятностное программирование использует-

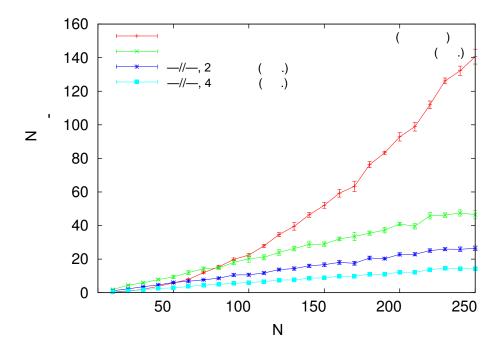


Рисунок 3 — Эффективность статистической вывода при его различных реализациях: красный график соответствует простейшему алгоритму, описанному в 1.2.5, при котором каждый раз происходит перевыполнение всей вероятностной программы. Зеленый график соответствует использованию новых алгоритмов и структур данных, что позволяет производить статистический вывод асимптотически более эффективно. Синий и голубой график показывают, что применение предлагаемого подхода также позволяет производить статистический вывод параллельно, по крайней мере приближенно. Рис. из [20].

ся для моделирования вероятностной модели находящихся на изображении объектов и их взаимодействия между собой. Байесовский подход к интерпретации изображений путем задания априорного распределения на расположение объектов и на связи между ними был предложен задолго до появления рассматриваемых языков программирования, однако именно с их появлением исследование и осуществление данного подхода стало проще, так как вероятностные языки программирования позволяют композиционно и компактно представлять вероятностные порождающие модели и проводить статистический вывод в них.

В работе рассматриваются два примера: проблема графической САРТСНА [23] — «компьютерного теста, используемого для определения, кем является пользователь системы: человеком или компьютером», знакомого почти что каждому пользователю интернета; и проблема нахождения на изображении с камеры автомобиля дороги, разделительной полосы, и левого и правого оврагов. Полученные результаты по своей эффективности на рассматриваемых простых примерах не уступают другим современным подходам к решению этих задач, однако представление, моделирование и вывод проще осуществляется с

помощью вероятностного программирования.

#### 1.5 О различиях между Church, Venture, Anglican

Целью данной главы, очевидно, не было подробное описание этих вероятностных языков, а только краткое введение в них. Заинтересованному читателю мы можем порекомендовать продолжить свое знакомство с вероятностным языком Church с [2, 24, 25, 16], Venture — [3], Anglican — [26, 4, 27].

Хотя многое объединяет эти вероятностные языки, в некоторых принципиальных вещах они различаются. Например, Venture на данный момент больше позиционируется как универсальная платформа, включающая в себя разные виды алгоритмов и методов статистического вывода с запроектированной возможностью добавлять новые с помощью использования базовых компонент и методов. В рамках работы над Anglican развиваются методы обобщенного вывода с использованием методов фильтрации частиц. Church, с другой стороны, позволяет производить статистический запрос внутри другого статистического запроса, что Venture и Anglican пока делать не могут.

### 2 Автоматическая генерация вероятностных программ

При использовании полных по Тьюрингу вероятностных языков программирования, включающих в себя функции высших порядков<sup>3</sup>, которыми в том числе являются языки Church, Venture и Anglican, вероятностная программа одновременно является и порождающей моделью, и записанной процедурой для генерации элементов выборки из этой модели путем выполнения исходного кода данной процедуры. Любая процедура в вероятностном программировании является формально программным кодом, который описывает процесс генерации элемента выборки при заданных аргументах данной функции. Таким образом, процедуры вероятностных программ являются конструктивным способом описания условных распределений.

Полные по Тьюрингу и допускающие функции высших порядков вероятностные языки программирования открывают возможность проведения вывода исходного текста самих вероятностных программ, если задано априорное распределение на множестве исходного текста, с помощью операторов eval и apply. Грубо говоря, необходимо представить вероятностную порождающую мета-модель, которая будет генерировать вероятностные модели в виде исходного кода вероятностных программ.

Данная глава основана на работе [6], которая включает в себя первые предварительные результаты по этой амбициозной задаче вывода самих порождающих вероятностных моделей при наличии какой-либо информации об искомом распределении, которое определяется искомой вероятностной программой. Отметим, что статистический вывод в пространстве исходного кода сложен и нет какого-то простого подхода как к построению вероятностных порождающих моделей исходного кода [28], так и выводу в них [29].

В рамках нашей предварительной работы мы поставили задачу найти с помощью статистического вывода и предлагаемой нами порождающей вероятностной мета-модели такие вероятностные программы, которые будут генерировать выборку элементов, схожую по своим статистическим характеристикам с каким-то заранее заданным распределением.

Эта задача интересна сама по себе, так как нахождение эффективных алгоритмов генерации (моделирования) случайных величин — нетривиальная задача и для людей-

<sup>&</sup>lt;sup>3</sup>Функциями (процедурами) высших порядков называются функции, аргументами или значениями которых могут быть другие функции.

ученых, которой они занимаются на протяжении десятков лет [30, 31, 32].

Наши предварительные результаты показывают, что подобный автоматизированный вывод порождающих вероятностных моделей в виде вероятностных программ действительно возможен. В частности, мы приводим результаты успешного эксперимента, в рамках которого мы автоматически нашли обобщенную вероятностную программу, которая подлинно (не приближенно) генерирует случайные величины, распределенные по закону Бернулли с произвольным параметром p.

#### 2.1 Обзор литературы

Рассматриваемые нами идеи относятся к разным областям, в том числе к автоматизации процесса программирования [33, 34], индуктивному программированию [33, 35, 36, 37, 38, 39], автоматическому моделированию [40, 41], компьютерному определению и представлению плотности распределений. Подходы к решению проблемы включают статистический вывод, поиск и оптимизацию, в том числе эволюционные алгоритмы и в особенности генетическое программирование.

Идеи и методы использования вероятностного программирования для изучения и автоматизированного представления вероятностных моделей предлагались и ранее [2, 42, 43, 3].

Насколько автору известно, описываемый подход к порождению вероятностных программам мета-вероятностной программой ранее не рассматривался в качестве отдельной проблемы достаточно подробно, хотя первые шаги в исследовании и формулировке проблемы были сделаны в работах [42, 3].

#### 2.2 Описание подхода

Наш подход может быть описан в рамках приближенных Байесовских вычислений [44] с использованием метода Монте-Карло по схеме цепей Маркова с выбором в качестве искомого апостериорного распределения

$$\pi(\mathcal{X}|\hat{\mathcal{X}})p(\hat{\mathcal{X}}|\mathcal{T})p(\mathcal{T}),\tag{1}$$

где  $\pi(\mathcal{X}|\hat{\mathcal{X}})$  — вероятностная мера, измеряющая расстояние между значения статистик, вычисленных соответственно на выборке  $\mathcal{X}$  из искомого распределения и выборке  $\hat{\mathcal{X}}$ , по-

```
[ASSUME program—text (productions '() 'real)]
[ASSUME program (eval (list 'lambda '() program—text))]
[ASSUME samples (apply—n—times program 10000 '())]
[OBSERVE (normal (mean samples) noise—level) 0.0]
[OBSERVE (normal (variance samples) noise—level) 1.0]
[OBSERVE (normal (skewness samples) noise—level) 0.0]
[OBSERVE (normal (kurtosis samples) noise—level) 0.0]
[PREDICT program—text]
[PREDICT (apply—n—times ... (program))]
```

Рисунок 4 — Вероятностная программа для вывода исходного кода вероятностной программы для генерации случайных чисел, распределенных по закону стандартного нормального распределения Normal(0,1).

лученной путем выполнения исходного кода вероятностной программы-кандидата.

Пусть существует распределение F с параметрами  $\theta$ , вероятностную программу для генерации элементов выборки из которого мы хотим вывести. Пусть  $\mathcal{X} = \{x_i\}_{i=1}^I, x_i \sim F(\cdot|\theta)$  будет выборкой из I элементов из распределения F при каком-то фиксированном значении параметров  $\theta$ . Рассмотрим задачу вывода исходного кода вероятностной программы  $\mathcal{T}$ , которая при ее неоднократном выполнении J раз сгенерирует выборку элементов  $\hat{\mathcal{X}} = \{\hat{x}_j\}_{j=1}^J, \hat{x}_j \sim \mathcal{T}(\cdot)$ , статистически близких к распределению F при заданных параметрах  $\theta$ .

Пусть s будет статистикой выборки, и тогда  $s(\mathcal{X})$  и  $s(\hat{\mathcal{X}})$  — значение этой статистики на элементах выборки из F и из  $\mathcal{T}(\cdot)$  соответственно. Пусть вероятностная мера  $d(s(\mathcal{X}),s(\hat{\mathcal{X}}))=\pi(\mathcal{X}|\hat{\mathcal{X}})$ , для простоты ненормированная, принимает бо́льшие значения, когда  $s(\mathcal{X})\approx s(\hat{\mathcal{X}})$ . d можно интерпретировать как расстояние или штраф.

Мы используем вероятностное программирование для представления мета-модели, порождающей другие вероятностные программы в виде исходного текста, и для проведения статистического вывода в пространстве искомых вероятностных моделей. Для статистического вывода мы использовали программную реализацию вероятностного языка программирования Anglican, которая поддерживает [4] статистический вывод методом частиц и методом Метрополиса-Гастингса по методу Монте-Карло по схеме цепей Маркова.

Вероятностная мета-модель представлена на рис. 4, где на первой строке мы устанавливаем соответствие между  $\mathcal{T}$  и переменной program-text, которая будет содержать один сгенерированный элемент из распределения на исходный код  $P(\mathcal{T})$ , определенное априорно через порождающую процедуру production с помощью адаптивной грамматики по типу [45] (см. подробнее в разделе 2.3). Мы не указываем здесь  $\theta$ , так как задача вывода в данном случае найти вероятностную программу, генерирующую элементы из cmandapmhoro нормального распределения. Переменная samples на второй строке представляет описанную выше выборку  $\hat{\mathcal{X}}$  из вероятностной программы-кандидата, и в этом примере J=10000.

s и d вычисляются на следующих четырех строках вероятностной программы, где статистика s определяется как четырехмерный вектор, включающий в себя соответственно выборочные среднее, дисперсию, коэффициент асимметрии и коэффициент эксцесса выборки элементов из распределения, определенного вероятностной программой, полученной из распределения  $\mathcal{T}$ . Мера расстояния d определяется через плотность многомерного нормального распределения со средними  $[0.0,1.0,0.0,0.0]^T$  и диагональной ковариационной матрицей  $\sigma^2\mathbf{I}$ . Отметим, что это означает, что мы ищем вероятностные программы, результат выполнения которых определяет распределение со средним равным 0, дисперсией -1, коэффициентами асимметрии и эксцесса равными 0, и мы «штрафуем» отклонения от этих значений с помощью квадратичной экспоненциальной функции потерь с коэффициентом  $\frac{1}{\sigma^2}$ , где  $\sigma$  определена как noise-level. Эта функция потерь представляет собой функцию плотности нормального распределения.

Данный пример служит хорошей иллюстрацией основных особенностей нашего подхода. Для поиска в виде вероятностной программы генератора случайных чисел из стандартного нормального распределения мы используем аналитическую информацию s(F) о стандартном нормальном распределении при вычислении расстояния между статистиками  $d(s(F), s(\hat{\mathcal{X}}))$ . Существует по крайней мере три различных ситуации, включая данную, в которых s и d могут вычисляться различными способами:

- 1. В рамках первой ситуации мы ищем вероятностную программу, которая эффективно генерирует элементы из аналитически известных распределений. Под эффективностью в данном случае можно понимать вычислительную временную сложность и среднее количество использованной энтропии для генерации элемента выборки в среднем. Практически всегда в этой ситуации и при подобной постановке задачи статистики распределения F известны аналитически.
- 2. Вторая ситуация возникает, когда мы можем генерировать только элементы выборки из F. Например, подобная ситуация возникает, когда мы генерируем элементы из апостериорного распределения с помощью «дорогостоящего» (вычис-

Рисунок 5 — Вероятностная программа для вывода исходного кода вероятностной программы, генерирующей случайные числа, распределенные по закону Бернулли Bernoulli( $\theta$ ). На предпоследней строчке выводится текст вероятностной программы-кандидата. На последней строчке вероятностная программа-кандидат выполняется Ј раз для генерации выборки из J элементов при параметризации p=0.3, причем предыдущие  $\theta_n$  (т.о. тренировочные значения параметров) не содержали p=0.3.

лительно) метода Монте-Карло по схеме Марковских цепей, и мы заинтересованы получить другое представление апостериорного распределения в виде вероятностной программы, чье априорное распределение будет точно или хотя бы приблизительно совпадать с искомым апостериорным.

3. При третьей ситуации нам заранее дана фиксированного размера выборка из F, и мы хотим найти вероятностную программу, которая позволит генерировать выборку произвольного размера из F в дальнейшем. Мы начали этот раздел с постановки задачи именно в рамках третьей ситуации.

Следует отметить, что возможная польза представления генератора случайных чисел потенциально не только в том, чтобы эффективно и быстро генерировать выборку из желаемого распределения, но и чтобы получить формальное представление желаемого распределения или его приближения в виде исходного кода вероятностной программы, то есть в виде формальной сущности, дальнейшие действия с которой можно производить; в том числе проводить анализ над выведенным исходным кодом и использовать найденные блоки исходного кода для решения других задач.

Рис. 5 иллюстрирует другое важное обобщение решение задачи, сформулированной в самом начале с апостериорным распределением (1).

При постановке задачи на вывод генератора случайных чисел, распределенных согласно стандартному нормальному распределению, мы не параметризовали искомое рас-

```
[ASSUME poisson (lambda (rate)
[ASSUME box-muller-normal
  (lambda (mean std)
                                              (begin (define L (exp (* -1 rate)))
                                                      (define inner-loop (lambda (k p)
   (+ mean (* std
      (* (cos (* 2 (* 3.14159
                                                     (if (< p L) (dec k)
      (uniform-continuous 0.0 1.0)))
                                                        (begin (define u
                                                                  (uniform-continuous 0 1))
      (\mathbf{sqrt} \ (* -2)
        (log (uniform-continuous 0.0 1.0)
                                                          (inner-loop (inc k) (* p u))))))
                                                     (inner-loop 1 (uniform-continuous 0 1))))]
          )))))))]
```

Рисунок 6 — Найденные и записанные людьми исходные коды вероятностных программ для (cnesa) общего нормального распределения  $Normal(\mu, \sigma)$  [31] и (cnpasa) распределения Ilmultipup Ilmultipup

пределение никаким образом, так как у стандартного нормального распределения нет параметров, т.е.  $\theta = \emptyset$ . В общем случае распределения, которые мы хотим представить в виде вероятностных программ, имеют нетривиальную параметризацию, и представляют собой по сути семейство распределений. Мы хотим найти вероятностную программу, входные аргументы которой как раз бы и являлись параметрами искомого распределения, таким образом, эта вероятностная программа позволяла бы генерировать случайные величины из всего семейства. Для наглядности рассмотрим алгоритм генерации случайных чисел, распределенных по закону нормального распределения, с помощью преобразования Бокса-Мюллера, представленный в виде вероятностной программы на рис. 6.

Эта вероятностная процедура параметризована двумя параметрами: средним и стандартным отклонением. Для постановки обобщенной задачи вывода параметризованных вероятностных программам нам необходимо изменить наше искомое апостериорное распределение, включив в него параметр  $\theta$ , параметризующий искомое распределение F:

$$\pi(\mathcal{X}|\hat{\mathcal{X}},\theta)p(\hat{\mathcal{X}}|\mathcal{T},\theta)p(\mathcal{T}|\theta)p(\theta).$$

Нам бы хотелось вывести вероятностную программу, которая смогла бы обобщить все возможные значения параметра  $\theta$ . С допущением, что если мы выберем конечное число N различных параметризаций  $\hat{\theta}_i$ , мы получим обобщение всего семейства распределений в виде вероятностной программы, мы формулируем нашу задачу в виде следующего приближения с использованием приближенных Байесовских вычислений в рамках метода

Монте-Карло по схеме цепей Маркова:

$$\frac{1}{N} \sum_{n=1}^{N} \pi(\mathcal{X}_n | \hat{\mathcal{X}}_n, \theta_n) p(\hat{\mathcal{X}}_n | \mathcal{T}, \theta_n) p(\mathcal{T} | \theta_n) \approx \int \pi(\mathcal{X} | \hat{\mathcal{X}}, \theta) p(\hat{\mathcal{X}} | \mathcal{T}, \theta) p(\mathcal{T} | \theta) p(\theta) d\theta.$$

Вероятностная программа для поиска параметризованной вероятностной программы, генерирующей случайные числа, распределенных по закону Бернулли Bernoulli( $\theta$ ), представлена на рис. 5, и наглядным образом иллюстрирует применение данного допущения и приближения. При выбранных N различных параметризациях параметра p распределения Бернулли мы каждый раз генерируем J элементов из вероятностной программыкандидата, аккумулируя расстояние (штраф) между искомым распределением и полученным распределением, представляющим вероятностную программу-кандидата. В каждом конкретном случае для  $\theta_i$  мы высчитываем: 1) расстояние с использованием статистики G-теста (более «современный» аналог критерия согласия Пирсона и соответствующей статистики) в виде

$$G_n = 2\sum_{i \in 0,1} \#[\hat{\mathcal{X}}_n = i] \ln\left(\frac{\#[\hat{\mathcal{X}}_n = i]}{\theta_n^i (1 - \theta_n)^{(1-i)} \cdot |\hat{\mathcal{X}}_n|}\right),\,$$

где  $\#[\hat{\mathcal{X}}_n=i]$  — количество элементов выборки из  $\hat{\mathcal{X}}_n$ , принимающих значение  $i;\,2)$  а также соответствующее p-значение с нулевой гипотезой, утверждающей, что элементы выборки  $\hat{\mathcal{X}}_n$  являются и элементами выборки из распределения Bernoulli( $\theta_n$ ). Так как распределение статистики G-теста приблизительно распределено по закону распределения хи-квадрат, т.е.  $G \sim \chi^2(1)$  в нашем примере, мы можем представить и находить расстояние d в данном случае путем вычисления вероятности ложного отклонения нулевой гипотезы  $H_0: \hat{\mathcal{X}}_n \sim \text{Bernoulli}(\theta_n)$ . Ложное отклонение нулевой гипотезы эквивалентно успеху в проведении испытания Бернулли с вероятностью успеха равной p-значению.

#### 2.3 Грамматика и порождающие правила

С учетом наличия в нашем распоряжении выразительного вероятностного языка программирования, допускающего функции высших порядков и полного по Тьюрингу, наше априорное распределение об исходном коде искомых вероятностных программ также достаточно выразительно. В общих чертах оно схоже с адаптивными грамматиками [45], используемыми в [29], но имеет отличия, в частности связанные с созданием сред с ло-

кальными переменными. В виде псевдокода наше априорное распределение может быть представлено следующим образом (символ  $\rightarrow$  означает «может перейти в»):

- 1. Выражение  $expr_{type}|env \rightarrow$  в имя переменной, случайно выбираемой из среды переменных env с типом type.
- 2. Выражение  $expr_{type}|env \to в$  случайную константу типа type. Константы различных типов (целочисленные, вещественные и т.д.) генерируются из отдельного для каждого типа type процесса Дирихле<sup>4</sup>  $DP_{type}(H_{type},\alpha)$ , где базовое распределение  $H_{type}$  само по себе в общем случае являются смесью нескольких распределений. Например, для констант вещественного типа мы используем смесь нормального распределения (normal 0 10), равномерного непрерывного распределения (uniform-continuous -100 100) и равномерного дискретного распределения из множества  $\{-1,0,1,\ldots\}$ .
- 3. Выражение  $expr_{type}|env \rightarrow (procedure_{type}\ expr_{arg\_1\_type}\ ...\ expr_{arg\_N\_type})$ , где процедура procedure является случайно выбираемой примитивной детерминированной или стохастической (не составной) процедурой, определенной заранее в глобальной среде, с типом возвращаемого значения type.
- 4. Выражение  $expr_{type}|env \rightarrow (compound\_procedure_{type}\ expr_{arg\_1\_type}\ ...\ expr_{arg\_N\_type})$ , где  $compound\_procedure_{type}$  является составной процедурой, также генерируемой в соответствии с процессом Дирихле  $DP_{type}(G_{type},\beta)$ , где базовое распределение  $G_{type}$  случайным образом генерирует составную процедуру с типом возвращаемого значения type и количеством входных аргументов, распределенным по закону Пуассона, где каждый входной аргумент имеет свой произвольный тип. Тело body составной процедуры генерируется также случайным образом согласно этим же порождающим правилам и грамматике, но с учетом введение локальной среды, включающей в себя входные аргументы процедуры.
- 5. Выражение  $expr_{type}|env \rightarrow (let \ [(gensym)\ expr_{real}]\ expr_{type}|env \cup gensym))$ , где  $env \cup gensym$  означает среду, дополненную переменной с именем (gensym) и со значением, вычисляемым в соответствии с генерируемым случайным образом выражением согласно этих же порождающих правил.
- 6. Выражение  $expr_{type}|env \rightarrow (if \ (expr_{bool}) \ expr_{type} \ expr_{type}).$
- 7. Выражение  $expr_{type}|env \rightarrow (recur\ expr_{arg\ 1}\ _{type}\ ...\ expr_{arg\ M}\ _{type})$ , таким образом

 $<sup>^4{\</sup>rm He}$ нужно путать с распределением Дирихле.

рекурсивный вызов текущей составной процедуры.

Во избежание вычислительных ошибок во время выполнения сгенерированных процедур мы заменяем примитивные функции их «защищенными» аналогами, например log(a) заменяется на safe-log(a), причем последний возвращает 0 если a < 0; или например uniform-continuous заменяется safe-uc(a, b), которая в случае если a > b меняет аргументы местами, а также возвращает просто a если аргументы равны: a = b.

Множество типов, которые мы использовали в рамках наших экспериментов, включало вещественные и булевы типы, а общее множество примитивных процедур, включенных нами в глобальную среду, включало в себя такие функции как +, -, \*, safe-div, safe-uc, safe-normal.

#### 2.4 Вероятности использования порождающих правил

Для задания априорных вероятностей порождающих правил, то есть вероятностей, с которой каждое из правил будет применяться в случае возможности его применения, мы вручную составили небольшой корпус вероятностных программ, которые повторяют найденные учеными [32] алгоритмы генераторов случайных чисел. Примеры таких программ представлены на рис. 6. Заметим, что все они требуют наличия только одной стохастической процедуры, а именно uniform-continuous, так что мы включили только ее в глобальную среду с положительной вероятностью для экспериментов, описанных в 2.5.2.

Используя данный корпус, мы вычислили априорные вероятности каждого порождающего правила, при этом при выводе вероятностной программы для генерации случайных величин из искомого распределения F (например, распределения Бернулли), мы исключали из корпуса все элементы, которые генерируют случайные величины согласно закону распределения F. После этого вероятности использования порождающих правил были «смягчены» с помощью распределения Дирихле. В будущем можно использовать более общирные корпусы вероятностных программ, примером зарождающегося подобного корпуса может служить [25].

#### 2.5 Эксперименты

Эксперименты были разработаны таким образом, чтобы проиллюстрировать все три вида возможных ситуаций, описанных ранее. Но в начале мы проиллюстрируем выразительность нашего априорного распределения исходного кода вероятностных программ. После

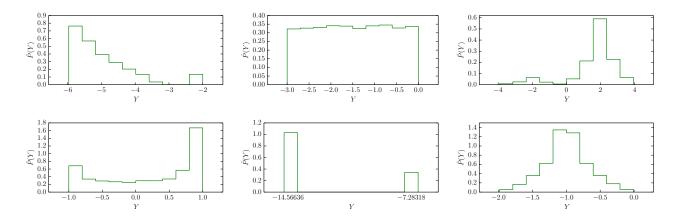


Рисунок 7 — Гистограммы выборок из некоторых порожденных вероятностных программ из их априорного распределения. По форме распределений видно, что наши порождающие правила достаточно обширны, чтобы соответствующие порождающим вероятностным программам распределения были нетривиальны.

этого мы опишем постановку и результаты экспериментов в рамках использования нашего подхода во всех трех различных ситуациях возможности вычисления расстояния d.

#### 2.5.1 Выборки из сгенерированных вероятностных программ

Для иллюстрации выразительности нашего априорного распределения исходных кодов вероятностных программ мы приводим выборки из случайным образом сгенерированных вероятностных программ из их априорного распределения. Эти шесть выборок в виде гистограмм из шести различных автоматически сгенерированных вероятностных программам расположены на рис. 7.

Из рисунка видно, что разные случайным образом сгенерированные вероятностные программы определяют в общем случае достаточно различные структурно распределения. В частности, можно заметить разнообразие в носителе, дисперсии и количестве «режимов» (т.е. островков носителя с высокой вероятностью по отношению к очень низковероятностным пространствам между островками).

# 2.5.2 Вывод вероятностных программ, определяющих распределения, совпадающие или приближающие классические одномерные распределения

Как уже было отмечено, для всех классических одномерных распределений существуют алгоритмы, позволяющие и точно, и приближенно генерировать любое количество элементов из данных параметризованных распределений, и они были аналитически выведены учеными. Эти алгоритмы в том числе, очевидно, могут быть записаны как вероятностные программы в виде их исходного кода.

Мы провели серию экспериментов, чтобы проверить возможность автоматического вывода вероятностных программ, генерирующих выборки из классических одномерных распределений, а именно из распределения Бернулли Bernoulli(p), распределения Пуассона Poisson $(\lambda)$ , Гамма-распределения Gamma(a,1.0), Бета-распределения Beta(a,1), стандартного нормального распределения Normal(0,1), и «общего» нормального распределения Normal $(\mu,\sigma)$ .

Для каждого семейства распределений мы проводили статистический вывод, выбирая в качестве целевого апостериорного распределения частное распределение  $\pi(\mathcal{X}|\hat{\mathcal{X}})p(\hat{\mathcal{X}}|\mathcal{T})p(\mathcal{T})$ , маргинализированное по параметру  $\theta$ . Для приближения в каждом случае мы выбирали малое множество значений параметров  $\{\theta_1,\ldots,\theta_N\}$  и определяли апостериорное распределение ограничениями по p—значению (для распределения Бернулли) и по близости моментам к ожидаемым (для всех других рассматриваемых распределений). Еще раз отметим, что при задании априорного распределения на порождающие правила в каждом конкретном случае из корпуса исключались все вероятностные программы, относящиеся к искомому распределению.

Образцы гистограмм выборок из лучших найденных в результате вывода вероятностных программ представлены на рис. 8, где под лучшими мы понимаем вероятностные программы с наименьшим расстоянием d на тренировочных значениях параметров и соответствующих значениях моментов или максимального p-значения.

Стоит особенно отметить результат эксперимента с распределением Бернулли, в рамках которого был найден исходный код вероятностной программы, статистически подлинно (точно) генерирующей выборку из всего семейства распределения Бернулли с параметром p. Найденная вероятностная программа представлена на рис. 9.

На рис. 10 представлен выведенный исходный текст вероятностной программы для генерации элементов из Гамма-распределения  $\operatorname{Gamma}(a,1)$ , параметризованного параметром a.

# 2.5.3 Обобщение произвольных эмпирических данных с помощью порождающего вероятностного программирования

Мы также проверили наш метод на выводе порождающих моделей в виде вероятностных программ для объяснения настоящих, не синтетических, данных, аналитическое распределение которых неизвестно.

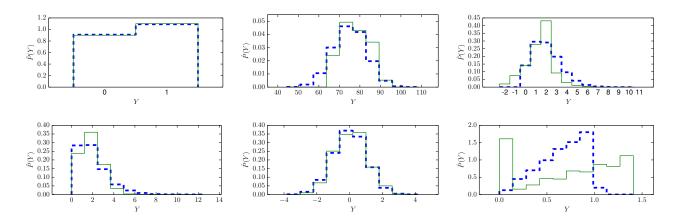


Рисунок 8 — (Зеленые сплошные линии) Гистограммы выборок из распределений, соответствующих вероятностным программам, имеющим «высокую» вероятность в апостериорном распределении при поиске порождающих моделей, соответствующих распределениям (слева направо, сверху вниз): Bernoulli(p), Normal(p, Normal(p, Poisson(p), Gamma(p, Normal(p), Beta(p, 1.0). (Синие пунктирные линии) Гистограммы выборок из «настоящих» распределений. Параметризация семейства распределений в каждом случае производилась другими значениями параметров, которые не входили в обучающее множество значений (p). Отметим, что для семейства распределений Бернулли Bernoulli(p) была выведена вероятностная программа, статистически точно генерирующая элементы выборки при любой параметризации (см. рис. 9). С другой стороны, не все распределения получили хорошее приближение найденными вероятностными программами за то время, что мы проводили вывод, как, например, в случае Бета-распределение Beta(p, 1.0).

```
(lambda (par stack-id) (if (< (uniform-continuous 0.0 1.0) par) 1.0 0.0)) (lambda (par stack-id) (if (< 1.0 (safe-sqrt (safe-div par (safe-uc par (dec par))))) 1.0 0.0)) (lambda (par stack-id) (if (< 1.0 (safe-uc (safe-sqrt par) (+ par (\cos par)))) 1.0 0.0))
```

Рисунок 9 — (сверху) Написанный человеком исходный код генератора случайных чисел, распределенных по закону Бернулли Bernoulli(p). (внизу, две программы) Выведенные исходные коды. Первая из двух выведенных вероятностных программ определяет настоящее семейство распределений Бернулли Bernoulli(p), параметризованное p. Вторая программа генерирует распределение, приближенное к распределению Бернулли, параметризованное p.

```
(lambda (par stack-id) (* (begin (define sym0 0.0)
                                                                   (lambda (stack-id)
    (exp (safe-uc -1.0 (safe-sqrt (safe-uc
                                                                     (* 2.0 (* (*
                                                                          (* -1.0 (safe-uc 0.0 2.0))
    (\,\mathrm{safe}\!-\!\mathrm{div}\ (\,\mathrm{safe}\!-\!\mathrm{uc}\ 0.0\ (\,\mathrm{safe}\!-\!\mathrm{uc}\ 0.0\ 3.14159))
      par) (+ 1.0 (safe-uc (begin (define sym2
                                                                          (safe-uc (safe-uc 4.0
      (lambda (var1 var2 stack-id) (dec var2)))
                                                                            (+ (safe-log 2.0) -1.0))
       (sym2 (safe-uc -2.0 (* (safe-uc 0.0 (begin )
                                                                            (* (safe-div 2.0
                                                                               -55.61617747203855)
       (define sym4 (safe-uc sym0 (* (+ (begin
       (define sym5 (lambda (var1 var2 stack-id)
                                                                              (if (< (safe-uc
       (safe-div (+ (safe-log (dec 0.0)) -1.0) var1)))
                                                                                (safe-uc
                                                                                27.396810474207317
       (sym5 (exp par) 1.0 0)) 1.0) 1.0)))
      (if (< (safe-uc par sym4) 1.0) sym0
                                                                                 (safe-uc -1.0 2.0))
      (safe-uc 0.0 -1.0))) sym0))
                                                                                2.0) 2.0)
       (safe-div sym0 (exp 1.0)) 0.0))))))))) par))
                                                                                4.0 (-1.0)))) -1.0)))
```

Рисунок 10 — Исходный код выведенных вероятностных программ для (слева) Гаммараспределения Gamma(a, 1) и (справа) для третьей эмпирической выборки индикаторов, используемых для рассмотрения заявок на выдачу кредита. Выборки, сгенерированные с помощью выполнения этих программ, расположены соответственно на рис. 8 и на рис. 11 (последняя гистограмма из трех). Для экономии места исходный код был сокращен, где возможно; например, путем замены (\* 1.0 0.0) на 0.0.

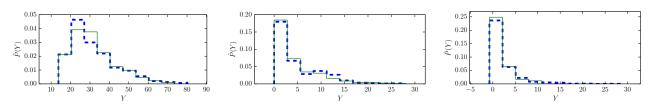


Рисунок 11 — Гистограммы (зеленые сплошные линии) выборок из распределений, определяемых найденными в процессе статистического вывода вероятностными программами для аппроксимации эмпирических данных (синие пунктирные линии) трех вещественных показателей из базы данных, используемой для анализа заявок на выдачу кредита.

Для данного эксперимента мы выбрали три набора данных признаков обращающихся в банк клиентов для получения кредита [47, 48], и производили вывод вероятностной программы, используя для сравнения *p*-значения двухвыборочного теста Колмогорова-Смирнова аналогично тому, как мы использовали G-тест для дискретных распределений. Гистограммы выборок из лучших найденных вероятностных программ в сравнении с гистограммами истинных эмпирическими данными признаков клиентов приведены на рис. 11. Пример выведенной вероятностной программы показан на рис. 10 *(справа)*.

#### 2.5.4 «Компиляция» вероятностных программ

Генерация выборок из апостериорного распределения методом Монте-Карло по схеме Марковских цепей, особенно в случае Байесовского вывода, обычно достаточно дорогостояща. Под «компиляцией» вероятностных программ мы имеем в виду поиск вероятностных программ, априорное распределение которых согласовывалось бы точно или приблизительно с искомым апостериорным распределением.

В вероятностном программировании, где в общем случае генерация выборки из же-

```
[ASSUME theta (beta 1.0 1.0)]
[OBSERVE (flip theta) True]
[PREDICT theta]
[PREDICT (flip theta)]
[PREDICT theta]
[PREDICT theta]
[PREDICT theta]
```

Рисунок 12-(cлева) Вероятностная модель бета-биномиального распределения (в несжатой форме) в виде вероятностной программы. В рамках эксперимента была поставлена задача найти формализацию апостериорного распределения скрытого параметра  $\theta$  в виде другой вероятностной программы, чье априорное распределение будет совпадать или приближать данное апостериорное. (cnpaвa, csepxy) Найденные вероятностные программы (т.е., результат «компиляции» вероятностных программ), априорное распределение которых приближает заданное апостериорное распределение. (cnpaвa, shusy) Записанный человеком исходный код вероятностной программы, чье априорное распределение совпадает с априорным распределением. В данном конкретном случае данная вероятностная программа может быть просто выведена аналитически, так как Бета-распределение является сопряженным к биномиальному, но в общем случае это нетривиальная задача.

лаемого апостериорного распределения путем использования методом Монте-Карло или аналогичным им является единственным доступным средством за разумное время, эта проблема стоит еще острее.

В качестве самых предварительных результатов мы провели эксперимент, в рамках которого наша априорная модель была моделью бета-биномиального распределения (в несжатой форме) с априорным распределением на скрытый параметр  $\theta \sim \text{Beta}(1.0, 1.0)$ , и использовали алгоритм Метрополиса-Гастингса для получения выборки  $\hat{\mathcal{X}}$  из апостериорного распределения скрытого параметра  $\theta$  с учетом проведения четырех успешных испытаний по схеме Бернулли Bernoulli( $\theta$ ). Соответствующая вероятностная программа представлена на рис. 12.

Затем мы использовали наш подход к выводу вероятностных программ, априорное распределение которых будет статистически схожим с полученной выборкой из желаемого апостериорного распределения. Примеры найденных вероятностных программ даны на рис. 12. В данном конкретном случае мы можем аналитически найти и записать в виде вероятностной программы апостериорное распределение, равное Beta(5.0, 1.0). Таким образом, полученные результаты показывают, что мы нашли хорошее приближение апостериорного распределения.

### 2.6 Обобщение порождающего вероятностного программирования

Как было отмечено ранее, порождающие вероятностные модели определяют совместное распределение (T,X), часто задаваемое сначала с помощью распределения (T), а затем с помощью условного распределения  $(X\mid T)$ . Порождающие вероятностные модели могут использоваться либо для генерации выборок напрямую из (T,X), либо в качестве промежуточного этапа для нахождения условного распределения скрытых параметров  $(T\mid X=\hat{x})$ , обычно с помощью приближенных методов статистического вывода или оптимизации.

В рамках наших экспериментов, описанных выше, мы решали задачу вывода вероятностной программы, которая определяет вероятностную модель (в общем случае параметризованную), априорное распределение которой соответствует или приближает искомое распределение, заданное аналитически, или в виде выборки, или в виде дорогостоящего генератора элементов выборки.

На языке вероятностного программирования Church, поддерживающего проведение статистического запроса внутри другого статистического запроса, данная задача может быть сформулирована следующим образом (подобно тому, как мы ее формулировали в Anglican/Venture, см. 4 и 5):

```
(query
2
     (define program-text (productions INPUT-TYPES OUTPUT-TYPE))
3
     (define program
        (eval (list 'lambda '(arg1 ... arg N) program-text)))
4
5
     program-text
6
     and
7
        (noisy-distance-equal
8
          (get-statistics
9
            (multiple-query
10
              ; No defines.
              (program arg1 1 ... arg1 N)
11
12
              true))
13
          expected-statistics1)
14
        . . .
```

где (query defines expression predicate) — задание запроса в Church (см. [2]), INPUT-TYPES — перечисление типов входных аргументов, OUTPUT-TYPE — тип выходного значения процедуры, noisy-distance-equal — функция сравнения статистик, get-statistics — функция извлечения статистик из распределения, multiple-query — аналог query, возвращающий несколько элементов из распределения вместо одного элемента, expected-statistics... — М значений статистик искомого распределения.

Автор считает, что более общо, для нахождения вероятностной модели, служащей вспомогательным инструментом в нахождении условного апостериорного распределения  $P(T\mid X=\hat{x})$ , мы можем быть заинтересованы в постановке задачи следующим образом:

```
1
   (query
2
     (define latent-variables-creator (latent-productions LATENT-TYPES))
     (define program-text
3
       (productions LATENT-TYPES INPUT-TYPES OUTPUT-TYPE))
4
     (define program
5
6
       (eval (list 'lambda '(arg1 ... argN) program-text)))
7
     (list latent-variables-creator program-text)
     (and
8
9
       (noisy-latents-equal
10
         (multiple-query
11
           (define latent-variables (latent-variables-creator))
12
           latent-variables
13
           (noisy-outputs-equal
             (program latent-variables arg1 1 ... arg1 N) output1))
14
```

```
expected-latents1)
15
16
        . . .
17
        (noisy-latents-equal
18
          (multiple-query
19
            (define latent-variables (latent-variables-creator))
            latent-variables
20
21
            (noisy-outputs-equal
              (program latent-variables argM 1 ... argM N) outputM))
22
23
          expected-latentsM)))
```

где latent-productions — правила порождения объекта, представляющего скрытые параметры типов LATENT-TYPES, «лениво» создаваемые с помощью процедуры mem (см. [2]); latent-variables — объект, представляющий скрытые параметры; arg...\_ ... — М значений аргументов, output... — М значений наблюдаемых данных; expected-latents... — М значений ожидаемых скрытых параметров.

Таким образом, в рамках данной формулировки мы ищем вероятностную программу, содержащую: скрытые переменные, определяемые latent-variables; наблюдаемые переменные, определяемые выполнением program; и вывод производится с учетом М тренировочных троек

$$\{a_i, \hat{t}_i, \hat{x}_i\},\$$

где  $a_i$  — дополнительная параметризация,  $\hat{x}_i$  — значения наблюдаемых переменных,  $\hat{t}_i$  — ожидаемые значения скрытых переменных.

Следует отметить, что так как в настоящее время обобщенный статистический вывод с использованием вероятностных языков программирования производится достаточно медленно, то решение практических задач таким образом пока не имеет смысла. С другой стороны, с дальнейшим развитием [4, 3] обобщенных методов статистического вывода и других решений (например, оптимизации или поиска), с использовании более продуманных алгоритмов программной реализации и структур данных [3, 27], со специализированным аппаратным обеспечением [49, 50], автоматизированная генерация и использование порождающих вероятностных моделей может успешно и продуктивно применяться на практике.

## ЗАКЛЮЧЕНИЕ

В первой части данной работы очень кратко и реферативно представлено введение в вероятностное программирование на примере языков Church/Venture/Anglican, первое подобное введение на русском языке, насколько известно автору.

Во второй части данной работы представлен подход к порождению вероятностных программ, обобщающих распределения, представленные либо в виде выборки, либо в виде аналитического представления (например, в виде значения статистик), причем метавероятностная модель вероятностных моделей также записывается в виде вероятностной программы. Используя данный подход, были получены предварительные положительные результаты статистического вывода из апостериорного распределения искомых вероятностных программ методами Монте-Карло по схеме цепей Маркова, в том числе была автоматически выведена вероятностная программа, генерирующая элементы выборки всего семейства распределений Бернулли.

### СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

- [1] Goodman Noah D. The principles and practice of probabilistic programming // ACM SIGPLAN Notices / ACM. Vol. 48. 2013. P. 399–402.
- [2] Church: A language for generative models / Noah D. Goodman, Vikash K. Mansinghka, Daniel M. Roy et al. 2008. P. 220–229.
- [3] Mansinghka Vikash, Selsam Daniel, Perov Yura. Venture: a higher-order probabilistic programming platform with programmable inference // arXiv preprint arXiv:1404.0099.— 2014.
- [4] Wood Frank, van de Meent Jan Willem, Mansinghka Vikash. A new approach to probabilistic programming inference // Proceedings of the 17th International conference on Artificial Intelligence and Statistics. 2014.
- [5] Approximate Bayesian image interpretation using generative probabilistic graphics programs / Vikash Mansinghka, Tejas D Kulkarni, Yura N Perov, Josh Tenenbaum // Advances in Neural Information Processing Systems. — 2013. — P. 1520–1528.
- [6] Perov Yura N, Wood Frank D. Learning probabilistic programs (статья отправлена на конференцию). 2014.
- [7] Лопатников Леонид Исидорович. Экономико-математический словарь. 5-е изд., перераб. и доп. М.: Дело, 2003.
- [8] Ветров Дмитрий Петрович, Кропотов Дмитрий Александрович. Байесовские методы машинного обучения (курс лекций). Методы Монте-Карло по схеме Марковских цепей. — 2011. — URL: http://www.machinelearning.ru/wiki/images/6/6b/BMM011\_ 10.pdf.
- [9] Рассел Стюарт, Норвиг Питер. Искусственный интеллект: современный подход (2-е издание). «Вильямс», 2007.
- [10] Bishop Christopher. Pattern Recognition and Machine Learning. Springer, New York, 2006.
- [11] Murphy Kevin P. Machine learning: a probabilistic perspective. MIT Press, 2012.
- [12] Ветров Дмитрий Петрович, Кропотов Дмитрий Александрович. Байесовские методы

- машинного обучения (курс лекций). 2013.- URL: http://www.machinelearning.ru/.
- [13] Сайт о вероятностном программировании.— 2014.— URL: http://probabilistic-programming.org/.
- [14] Абельсон Харольд, Сассман Джеральд Джей. Структура и интерпретация компьютерных программ. Добросвет, КДУ, 2010.
- [15] Freer Cameron, Mansinghka Vikash, Roy Daniel. When are probabilistic programs probably computationally tractable? // NIPS 2010 Workshop on Monte Carlo Methods in Modern Applications. 2010.
- [16] Wingate David, Stuhlmueller Andreas, Goodman Noah D. Lightweight implementations of probabilistic programming languages via transformational compilation. 2011. P. 131.
- [17] Николенко Сергей Игоревич. Вероятностное обучение (курс лекций). 2007. URL: http://logic.pdmi.ras.ru/~sergey/index.php?page=mlbayes.
- [18] Chib Siddhartha, Greenberg Edward. Understanding the metropolis-hastings algorithm // The American Statistician. 1995. Vol. 49, no. 4. P. 327–335.
- [19] Зубков А. М., Шуваев Д. В. Вычисление моментов комбинаторных статистик от перестановочных случайных величин // Дискретная математика. 2005.
- [20] Perov Yura, Mansinghka Vikash. Exploiting Conditional Independence for Efficient, Automatic Multicore Inference for Church. -2012.
- [21] Blei David M, Ng Andrew Y, Jordan Michael I. Latent dirichlet allocation // the Journal of machine Learning research. -2003.- Vol. 3.- P. 993-1022.
- [22] Wu Jeff. Reduced traces and JITing in Church : Ph. D. thesis / Jeff Wu ; Massachusetts Institute of Technology. 2013.
- [23] CAPTCHA: Using hard AI problems for security / Luis Von Ahn, Manuel Blum, Nicholas J Hopper, John Langford // Advances in Cryptology—EUROCRYPT 2003. — Springer, 2003. — P. 294–311.
- [24] Goodman N. D., Tenenbaum J. B. Probabilistic Models of Cognition.— 2014.— URL: https://probmods.org/.

- [25] A Repository for Generative Models (composite authors, edited by Andreas Stuhlmüller). —
  2014. URL: http://forestdb.org/.
- [26] Wood Frank et al. Probabilistic Programming Tutorial (Machine Learning Summer School, Iceland). — 2014. — URL: http://www.robots.ox.ac.uk/~fwood/anglican/teaching/ mlss2014/.
- [27] Paige Brooks, Wood Frank. A Compilation Target for Probabilistic Programming Languages. 2014.
- [28] Maddison Chris J, Tarlow Daniel. Structured Generative Models of Natural Source Code. 2014.
- [29] Liang Percy, Jordan Michael I, Klein Dan. Learning programs: A hierarchical Bayesian approach. — 2010. — P. 639–646.
- [30] Marsaglia George, Bray Thomas A. A convenient method for generating normal variables. -1964. Vol. 6, no. 3. P. 260–264.
- [31] Box George EP, Muller Mervin E et al. A note on the generation of random normal deviates. 1958. Vol. 29, no. 2. P. 610–611.
- [32] Devroye Luc. Non-Uniform Random Variate Generation. 1986.
- [33] Gulwani Sumit, Kitzelmann Emanuel, Schmid Ute. Approaches and Applications of Inductive Programming (Dagstuhl Seminar 13502).— 2014.— Vol. 3, no. 12.— P. 43—66.—URL: http://drops.dagstuhl.de/opus/volltexte/2014/4507.
- [34] Looks Moshe. Program evolution for general intelligence // FRONTIERS IN ARTIFICIAL INTELLIGENCE AND APPLICATIONS. -2007. Vol. 157. P. 125.
- [35] Muggleton Stephen, De Raedt Luc. Inductive logic programming: Theory and methods. 1994. — Vol. 19. — P. 629–679.
- [36] Muggleton Stephen, Feng Cao. Efficient induction of logic programs. 1992. Vol. 38. P. 281–298.
- [37] De Raedt Luc, Kersting Kristian. Probabilistic inductive logic programming. Springer, 2008.
- [38] Kersting Kristian. An inductive logic programming approach to statistical relational

- learning / IOS Press. 2005. P. 1-228. URL: http://people.csail.mit.edu/kersting/FAIAilpsrl/.
- [39] Muggleton Stephen. Stochastic logic programs. 1996. Vol. 32. P. 254–264.
- [40] Exploiting compositionality to explore a large space of model structures / Roger Grosse, Ruslan R Salakhutdinov, William T Freeman, Joshua B Tenenbaum. 2012.
- [41] Structure discovery in nonparametric regression through compositional kernel search / David Duvenaud, James Robert Lloyd, Roger Grosse et al. 2013.
- [42] Hwang Irvin, Stuhlmüller Andreas, Goodman Noah D. Inducing probabilistic programs by Bayesian program merging. 2011.
- [43] Gordon Andrew D. An Agenda for Probabilistic Programming: Usable, Portable, and Ubiquitous // Workshop on Probabilistic Programming: Democratizing Machine Learning. -2013.
- [44] Markov chain Monte Carlo without likelihoods / Paul Marjoram, John Molitor, Vincent Plagnol, Simon Tavaré. 2003. Vol. 100, no. 26. P. 15324–15328.
- [45] Johnson Mark, Griffiths Thomas L, Goldwater Sharon. Adaptor grammars: A framework for specifying compositional nonparametric Bayesian models. 2007. Vol. 19. P. 641.
- [46] Knuth Donald E. The Art of Computer Programming, 3rd edn., vol. 2. 1998.
- [47] Quinlan J. Ross. Simplifying decision trees. 1987. Vol. 27, no. 3. P. 221–234.
- [48] Bache K., Lichman M. UCI Machine Learning Repository.— 2013.— URL: http://archive.ics.uci.edu/ml.
- [49] Tenenbaum Joshua B, Jonas Eric M, Mansinghka Vikash K. Stochastic Digital Circuits for Probabilistic Inference. — 2008.
- [50] Jonas Eric Michael. Stochastic architectures for probabilistic computation: Ph. D. thesis / Eric Michael Jonas; Massachusetts Institute of Technology. — 2014.