Оглавление

В данном пособии предложены конспекты лекций по некоторым вопросам математического и имитационного моделирования информационных систем и сетей. Рассматриваются общие вопросы технологии имитационного моделирования, прежде всего различные способы представления систем с дискретными событиями, вопросы построения генераторов псевдослучайных объектов (в том числе графов и строк) и диагностики ошибок в программах имитационных моделей. Рассматриваются вопросы расчета вероятности связности случайных графов.

Издано на средства .....
Отпечатано .....
Тираж 250 экз.

Введение

В предлагаемом пособии представлены избранные лекции по имитационному моделированию систем с дискретными событиями, в которых в основном рассматриваются вопросы, слабо освещённые в отечественной учебной литературе. Исключение составляют первые лекции, содержащие базовые понятия и описывающие основные алгоритмы генерации независимых, одинаково распределённых случайных величин. Остальные лекции представляют собой либо изложение неизданного на русском языке материала (DEVS-схемы, графы событий), либо собственные результаты автора, опубликованные в разное время и в разных изданиях, в основном в виде журнальных статей и материалов конференций.

В полном объёме курс имитационного моделирования включает также ознакомление с одной или несколькими системами моделирования (обычно GPSS). Эта часть курса достаточно хорошо поддержана существующей литературой и Интернет-пособиями и справочниками и по этой причине, равно как в силу ограниченности заявленного объёма пособия, здесь не излагается. То же самое относится и к таким известным способам описания систем с дискретными событиями как сети очередей и агрегированные системы.

Каждая глава пособия содержит изложение одной или нескольких лекций. Рекомендации по разбиению материала по часам приводятся в начале каждой главы. Там же указываются ссылки на использованную литературу.

Представленный материал использовался и используется в практике преподавания курсов "Имитационное моделирование на ЭВМ "Имитационное моделирование вычислительных систем"и "Моделирование на ЭВМ"в Новосибирском государственном университете (НГУ), Высшем колледже информатики при НГУ и Сибирском государственном университете телекоммуникаций и информатики (СибГУТИ).

Глава 1

Лекция 1. Основные понятия и определения

Рекомендуемое время – 2 часа на все разделы.
Основные положения можно найти в [3, 1, 8, 2].

1.1 Основные типы моделей

В практике математического моделирования принято различать аналитические, численные и имитационные модели. Кратко их отличительные свойства можно определить следующим образом.

- **Аналитические модели**. Под такими моделями понимают описание процессов и явлений с помощью математических формул, например с помощью систем дифференциальных уравнений. Отличительной особенностью таких моделей является четкая интерпретация всех входящих в них переменных и параметров. Наличие аналитической модели в общем случае не предполагает наличия метода ее решения, то есть получения значений наблюдаемых переменных для заданных значений изменяемых переменных.

  Примером аналитической модели может являться уравнение колебания струны (одномерное волновое уравнение):

  $$\frac{d^2u}{dt^2} = a^2\frac{d^2u}{dx^2} + \frac{1}{\rho}g(x,t) \tag{1.1}$$

- **Численные модели**. Численная модель есть описание метода (алгоритма) решения модели аналитической. При решении задачи могут возникнуть ситуации, когда промежуточные результаты не интерпретируются в исходных терминах модели (например, при использовании метода квадратного корня при решении системы линейных уравнений с симметрической матрицей, исходная матрица системы представляется в виде $A = LL'$, где L – нижняя треугольная матрица, а L' – матрица, транспонированная по отношению к L. Очевидно, что невозможно содержательно интерпретировать элементы этих матриц).

- **Имитационные модели**. Эти модели представляют собою описание поведения системы (сценарий), по которому из одного состояния модели (в момент времени t_i) можно получить следующее состояние (в момент времени t_{i+1}), при этом и сценарий, и состояние (то есть промежуточные данные) полностью описываются и интерпретируются в исходных терминах.

Под **имитационным моделированием** в общем случае понимается построение имитационной модели системы и экспериментальное исследование этой модели на ЭВМ. Построение модели включает в себя разработку математико-логического представления системы и соответствующей программы, позволяющей модели имитировать поведение системы.

1.2 Основные понятия имитационного моделирования

Введем некоторые определения:

система – объединение сущностей, которые взаимодействуют друг с другом во времени для выполнения множества целей;

модель – математико-логическое представление системы в терминах сущностей и их атрибутов, наборов, событий, активностей и ожиданий;

состояние системы – набор переменных, значения которых определяют состояние системы в данный момент времени;

сущность – объект, единица или компонента системы, требующие точного представления в модели;

атрибут – свойства или характеристики данной сущности;

набор – множество взаимосвязанных сущностей;

событие – происходит мгновенно во времени, во время события происходит изменение состояния системы;

активность – отрезок времени, во время которого одна или более сущностей вовлечены в выполнение некоторой функции системы и момент окончания которого известен заранее;

ожидание – отрезок времени неопределенной длины, протяжение которого неизвестно до его окончания.

1.3 Этапы имитационного моделирования

В общем случае процесс имитационного моделирования включает:

1. **Формулировку задачи**, включая общее описание системы и определение ее границ.
2. **Определение целей** исследования, - определение вопросов, на которые должна ответить имитация.
3. **Построение модели**, - описание существенных (для данного исследования) свойств системы в терминах сущностей, атрибутов или характеристик каждой сущности, активностей, в которые вовлечены эти сущности и описание множества возможных состояний модели.
4. **Сбор информации** - сбор данных и информации, позволяющих разработчику модели разработку описания свойств сущностей и определения распределений вероятностей для параметров системы.
5. **Кодирование** - процесс перевода описания модели системы в программу для ЭВМ.
6. **Верификацию** - процесс подтверждения правильности функционирования программы (программа работает по своей спецификации).
7. **Валидацию** - процесс подтверждения адекватности модели (модель верно имитирует поведение моделируемой системы). Итерационные шаги валидации и верификации составляют основную часть разработки компьютерной программы.
8. **Планирование эксперимента** - определение альтернатив, которые должны быть рассмотрены в ходе имитационных экспериментов, включает выбор существенных входных переменных, определение их подходящих значений, определение длин экспериментов и их количества.
9. **Рабочие эксперименты** и анализ результатов.
10. **Документирование** программы и результатов моделирования.

1.4 способы представления систем в имитационных моделях

На этапе построения имитационных моделей дискретных систем необходимо выбрать подход к описанию событий и взаимосвязей компонент системы. При этом можно выбирать различные варианты формализации.

1.4.1 Формальные модели систем

При использовании формальных математических моделей для описания ИТС и их компонентов мы имеем следующие преимущества:

1. Проще сделать имитирующую программу, так как она должна обрабатывать одно универсальное описание системы и стандартно представленные данные. Изменения в модели приводят только к изменению входных данных;

2. Компоненты ИТС описываются для хранения стандартным образом (например, системы массового обслуживания в обозначениях Кендэла). Однако, при использовании подобных моделей мы имеем и ряд недостатков формального описания:

   - Формальное описание может привести к потере некоторых свойств моделируемого объекта;
   - Трудно давать задание на сбор нестандартной (если понадобится) статистики по поведению модели;

3. Необходимо давать интерпретацию результатов, поскольку они получаются в форме, соответствующей принятой формальной модели.

1.4.2 Общие схемы описания моделей систем

Под общей схемой описания модели системы понимаем её описание в рамках заданного способа описания её поведения: как совокупность взаимодействующих процессов, систему транзакций, систему событий и т.п. Принятый способ описания определяет, прежде всего, способ программного описания модели и состав структур данных и процедур. При использовании общих схем описания моделей систем с дискретными событиями наибольшую трудность представляет решение вопроса о хранении стандартных компонентов. Поведение компонентов описывается процедурами или процессами, соответственно организуются библиотеки стандартных программ. Изменения в модели приводят к необходимости изменения программы и её перекомпиляции. Несомненным достоинством использования общих схем описания моделей является гибкость такого подхода, возможность быстрого введения в систему моделирования новых алгоритмов управления системой и её компонентами. Для различного типа систем подходят разные схемы описания. Так, общепризнано, что из всех общих схем описания моделей систем с дискретными событиями для описания моделей вычислительных систем и сетей наиболее подходит транзактно-ориентированный подход (или сетевой, согласно терминологии системы моделирования СЛАМ-II [2]). Этот подход также является наиболее подходящим для описания поведения стандартных компонентов в форме подходящей для хранения, так как модель представляется в виде блок-схемы специального вида. Однако необходимо организовывать настройку параметров соответствующих блоков.

1.4.3 Формальные схемы описания систем

Под формальными схемами описания моделей систем с дискретными событиями (в отличие от общих схем) мы понимаем такие способы формализации, в которых строгость внешней формы сочетается со свободой описания правил поведения. В качестве примеров можно привести агрегированные системы, предложенные Н.П. Бусленко [3] и схожие с ними, но более современные и получившие более широкое распространение в настоящее время схемы DEVS, предложенные Б. Зайглером (B.

Zeigler) [4]. Подобные схемы обладают преимуществами формальных моделей в силу строгости и единообразия описания компонентов и в то же время позволяют описывать произвольные правила их поведения.

Глава 2

Лекции 2-5. Генерация псевдослучайных объектов.

Рекомендуемое время – по 2 часа на разделы с 2.1 по 2.2.1, разделы с 2.2.1 по 2.3,разделы 2.4 и 2.5, разделы с 2.6.1 по 2.6.2 и разделы с 2.6.3 по 2.6.3, соответственно.
Основные положения можно найти в [3, 1, 8, 2].

При описании поведения, а иногда и структур сложных систем часто приходится иметь дело со случайными объектами, в частном случае случайными числами и процессами. При их моделировании на ЭВМ, если не использовать специальные устройства, используются алгоритмы получения последовательностей величин которые, не являясь на самом деле случайными, *ведут себя как случайные*. Такие величины или объекты называются псевдослучайными.

В основе всех алгоритмов получения псевдослучайных объектов лежит использование так называемого *базового датчика случайных чисел (д.с.ч.)*, то есть генератора псевдослучайных чисел, равномерно распределенных на отрезке $[0,1]$ (реально, как правило, $[0,1)$). Далее такие числа обозначаются как ξ или ξ_i, в зависимости от контекста.

2.1 Базовый д.с.ч.

Вопросу построения качественного д.с.ч. посвящено достаточно много исследований. В основном этот вопрос обсуждается в литературе по так называемому методу "Монте-Карло"(см., например, [6], [7]). Достаточно строго и, вместе с тем, доступно этот вопрос обсужден в [8]. На основе опыта и анализа алгоритмов большинство авторов пришло к выводу о преимуществе использования так называемого линейного конгруэнтного метода. Линейная конгруэнтная последовательность имеет вид

$$u_{i+1} = (au_i + b) \bmod M \tag{2.1}$$

Данная последовательность очевидно периодична. Величина периода является одной из основных характеристик качества генератора. Приведем без доказательства следующую известную теорему (доказано М. Гринбергом для частного случая в 1961 и А. Добеллом и Т. Халлом в 1962 году для общего случая):

Длина периода линейной конгруэнтной последовательности равна M тогда и только тогда, когда

i) a и b – взаимно простые числа;

ii) $c = a - 1$ кратно p для любого простого p, являющегося делителем M;

iii) c кратно 4 если M кратно 4.

Однако, максимальный период ещё вовсе не означает хорошее качество датчика. Так, значения $a = 1$ и $b = 1$ (они взаимно просты, так как не имеют иных общих делителей кроме 1) при $u_1 = 0$ для любого M дадут полный период со значениями $1, 2, 3, \ldots, M-1$. Эту последовательность никак нельзя назвать похожей на случайную.

Таким образом, задача выбора коэффициентов метода нетривиальна и решается, как правило, экспериментальным путём. Общей рекомендацией является достаточно большое значение a в то время как b может быть и равным 0.

Далее всегда предполагается наличие качественного базового д.с.ч.

2.2 Генерация независимых, одинаково распределенных случайных величин

2.2.1 Непрерывные распределения

Существует несколько методов генерации независимых с.в. с заданным законом распределения. Наиболее точные из них основаны на преобразовании с.в. Так, большое количество датчиков получается исходя из известного результата о равномерном на $[0, 1)$ распределении функции $F_\eta(\eta)$, где η - произвольная непрерывная случайная величина с функцией распределения $F_\eta(x)$.

Пример 2.1. Требуется получить датчик экспоненциально распределенных с.в. В этом случае

$$F_\eta(x) = 1 - e^{-\lambda x} \tag{2.2}$$

Тогда из решения уравнения

$$\xi = 1 - e^{-\lambda \eta} \tag{2.3}$$

получаем $\eta = -\frac{\log(1-\xi)}{\lambda}$.

Можно заметить, что выражение под знаком логарифма в последней формуле имеет равномерное распределение на отрезке [0,1), что позволяет получить окончательный вид формулы генератора: $\eta = -\frac{\log \xi}{\lambda}$.

Генератор нормальных с.в.

Нормально распределенная с.в. ν имеет функцию плотности распределения вероятности

$$\phi(x) = \frac{1}{\sqrt{2\pi}\sigma} e^{(-\frac{x-a}{\sigma})^2}. \tag{2.4}$$

Связанная с нею нормализованная величина $u = \frac{\nu - a}{\sigma}$ имеет так называемое стандартное нормальное распределение с нулевым математическим ожиданием и единичной дисперсией. Далее речь идет о генерации именно таких с.в.

Для генерации нормально распределенных с.в. метод обратной функции напрямую не применим, поскольку для обратной функции нет аналитического выражения. Возможно приблизить обратную функцию по известным табличным значениям, но достижение хорошей точности потребует достаточно высокой степени аппроксимирующих полиномов. На практике используют кусочную аппроксимацию (три участка) дробно-полиномиальной функцией или используют другие методы. Наиболее простым из точных является метод полярных координат, при котором с использованием пары значений базового датчика получают пару независимых нормально распределенных с.в. Соответствующий алгоритм имеет следующий вид [8]:

Шаг 1. Получить два независимых случайных числа ξ_1 и ξ_2. Положим $v_1 = 2\xi_1 - 1$ и $v_2 = 2\xi_2 - 1$, эти величины равномерно распределены на (-1,1). Величины v_1 и v_2 лучше представить в машине в форме с плавающей запятой.

Шаг 2. Вычислить $S = v_1^2 + v_2^2$.

Шаг 3. Если $S \geq 1$ то повторить шаг 2.

Шаг 4. Присвоить величинам u_1 и u_2 значения, вычисленные по формулам

$$u_1 = v_1\sqrt{\frac{-2\log S}{S}} \quad u_2 = v_2\sqrt{\frac{-2\log S}{S}} \tag{2.5}$$

Другие датчики могут основываться на известных соотношениях между распределениями.

Пример 2.2. Генератор логарифмически нормальных с.в.

По определению, логарифмически нормальной с.в. называется с.в., логарифм которой распределен по нормальному распределению. Соответственно, если иметь генератор независимых нормально распределенных с.в. u(см. выше), требуемый генератор получается по формуле $\eta = e^u$.

Для построения приближенных генераторов можно использовать предельные теоремы теории вероятностей.

Пример 2.3. Генератор нормальных с.в.

Центральная предельная теорема теории вероятностей гласит "распределение суммы независимых, одинаково распределенных случайных величин сходится к нормальному с математическим ожиданием, равным сумме математических ожиданий и дисперсией, равной сумме дисперсий".

Соответственно, при достаточно большом n сумма $\sum_{i=1}^{n}\xi_i$ имеет нормальное распределение. Поскольку $E[\xi] = \frac{1}{2}$ и $D[\xi] = \frac{1}{12}$, удобно ограничить сумму двенадцатью слагаемыми:

$$\tilde{u} = \sum_{i=1}^{12}\xi_i - 6. \tag{2.6}$$

$\tilde{u}$ распределена на интервале (-6,6) и абсолютное отклонение функции плотности вероятностей от истиной не превышает 2%.

Метод отбраковки

В некоторых случаях требуется точное соответствие заданному закону распределения при отсутствии эффективных методов генерации. В такой ситуации для ограниченных с.в. можно использовать следующий метод. Функция плотности распределения вероятностей с.в. $f_\eta(x)$ вписывается в прямоугольник $(a,b)\times(0,c)$, такой, что a и b соответствуют границам диапазона изменения с.в. η, а c – максимальному значению функции плотности её распределения. Тогда очередная реализация с.в. определяется по следующему алгоритму:

Шаг 1. Получить два независимых случайных числа ξ_1 и ξ_2.

Шаг 2. Если $f_\eta(a+(b-a)\xi_1) < c\xi_2$ то выдать $a+(b-a)\xi_1$ в качестве результата. Иначе повторить Шаг 1.

Неэффективность алгоритма для распределений с длинными “хвостами” очевидна (см. Рис. 2.2.1), поскольку в этом случае часты повторные испытания. [h!tb](100,52)(0.00,0.00)select Случаи эффективного(a) и не эффективного (b) применения метода отбраковки

Случай кусочно заданной плотности распределения

Пусть случайная величина η определена на интервале $[a,b)$ и её плотность имеет вид (2.7)

$$\text{[h!tb] (79,41)(0.00,0.00)piece}_d ens f(x) = \begin{cases} f_1(x), & \text{при } a = x_0 \le x < x_1, \\ f_2(x), & \text{при } x_1 \le x < x_2, \\ \dots & \\ f_k(x), & \text{при } x_{k-1} \le x < x_k = b, \end{cases} \tag{2.7}$$

где $\forall i \in (0,\dots,k-1) x_i < x_{i+1}$. Обозначив $p_i = \int\limits_{x_{i-1}}^{x_i} f_i(x)dx$ имеем $\sum_{i=1}^{k} p_i = 1$.

Для получения значения η необходимо *сначала* выбрать плотность $f_i(x)$ с вероятностью p_i и *затем* выбрать случайное значение на интервале $[x_{i-1}, x_i)$ согласно выбранному распределению.

Использование линейных преобразований

Часто являются полезными следующие очевидные выражения для плотностей линейных функций случайной величины:

$\eta = a\nu + b \longrightarrow f_\eta(x) = f_\nu(\frac{x-b}{a}), \eta = -\nu \longrightarrow f_\eta(x) = f_\nu(-x)$.

Пример 2.4. Генератор треугольного распределения Пусть плотность распределения имеет вид, представленный на рисунке **??**

Вариант 1. Прямое применение метода обратной функции.

Сначала получим формулу функции распределения вероятностей как интеграл от плотности распределения:

$$F(x) = \begin{cases} 0, & \text{при } x < -1, \\ (x+1)^2/2, & \text{при } -1 \le x < 0, \\ 1-(1-x)^2/2, & \text{при } 0 \le x < 1, \\ 1, & \text{при } x \ge 1. \end{cases} \tag{2.8}$$

Тогда, решая уравнение $\xi = F^{-1}(\eta)$ относительно η имеем

$$\eta = \begin{cases} \sqrt{2\xi} - 1, & \text{при } \xi < 0.5, \\ 1 - \sqrt{2(1-\xi)}, & \text{при } \xi \ge 0.5. \end{cases} \tag{2.9}$$

Вариант 2. Кусочное рассмотрение функции распределения.

Область изменения рассматриваемой случайной величины можно разбить на 2 равных интервала $[-1, 0)$ и $[0, 1)$, на которые она попадает с равной вероятностью 0.5. Соответствующие плотности усечённых распределений на этих интервалах имеют вид

$$f_1(x) = 2(x+1), \tag{2.10}$$
$$f_2(x) = 2(1-x), \tag{2.11}$$

а функции —

$$F_1(x) = (x+1)^2, \tag{2.12}$$
$$F_2(x) = 1-(1-x)^2. \tag{2.13}$$

Следовательно, генератор должен с вероятностью 0.5 выдавать $\eta = \sqrt{\xi} - 1$ и с той же вероятностью $\eta = 1 - \sqrt{1-\xi}$.

Отметим, что в последней формуле можно заменить $1 - \xi$ на ξ, так как эти случайные величины имеют одинаковое распределение.

Вариант 3. Использование функции двух случайных величин.

Рассмотрим распределение суммы двух независимых, равномерно распределённых на отрезке $[0, 1)$ случайных величин. Пусть $\eta = \xi_1 + \xi_2$. Тогда $F_\eta(x) = P(\eta < x) = P(\xi_1 + \xi_2 < x)$. Удобнее всего получить эту вероятность геометрически (Рис. 2.2.1). Ей соответствует площадь части единичного квадрата, лежащей ниже прямой, проходящей через точки $(0, x)$ и $(x, 0)$. Для $x < 1$ (двойная штриховка) это $x^2/2$, а для $x \ge 1$ — $1 - (2-x)^2/2$. Следовательно,

[h!tb](85,71)(0.00,0.00)sum2xi Иллюстрация к выводу распределения суммы двух случайных величин, равномерно распределённых на $[0, 1)$

$$F(x) = \begin{cases} 0, & \text{при } x < 0, \\ x^2/2, & \text{при } 0 \le x < 1, \\ 1-(2-x)^2/2, & \text{при } 1 \le x < 2, \\ 1, & \text{при } x \ge 2. \end{cases} \tag{2.14}$$

Плотность распределения, соответственно, равна

$$f(x) = \begin{cases} 0, & \text{при } x < 0, \\ x, & \text{при } 0 \le x < 1, \\ 2 - x, & \text{при } 1 \le x < 2, \\ 0, & \text{при } x \ge 2, \end{cases} \tag{2.15}$$

то есть мы имеем ту же самую треугольную плотность распределения, но сдвинутую по оси абцисс на одну единицу вправо. Следовательно имеем $\eta = \xi_1 + \xi_2 - 1$!

2.2.2 Дискретные распределения

Наиболее общий случай построения генератора дискретных случайных величин основывается на следующем алгоритме. Пусть имеется таблица пар (x_i, p_i), где x_i – значение случайной величины, а p_i – соответствующая вероятность, $\sum_{i=1}^{n} p_i = 1$ (в общем случае может быть и $n = \infty$). Тогда последнюю сумму можно представить полуинтервалом $[0, 1)$, разбитым на полуинтервалы $[v_{i-1}, v_i)$, $v_0 = 0$, $v_n = 1$ длины p_i. Случайная величина ξ с неизбежностью принадлежит одному из этих полуинтервалов, тем самым определяя индекс дискретного значения. Номер полуинтервала очевидно определяется как

$$\min\{i \mid \xi < v_i\} = \min\{i \mid \xi < \sum_{j=1}^{i} p_j\} \tag{2.16}$$

Замечание. Для эффективности датчика рекомендуется один раз рассчитать все суммы в 2.16 и затем использовать *вектор накопленных вероятностей*.

Пример 2.4. Равномерное дискретное распределение.

В случае равновероятного выбора между $x_i = i$, $i = 1, \ldots, n$ все p_i равны $\frac{1}{n}$ и тем самым $v_i = \frac{i}{n}$. Соответственно имеем номер полуинтервала равным $\min\{i \mid \xi < \frac{i}{n}\}$, откуда получаем просто $i = \lceil n\xi \rceil$.

Пример 2.5. Распределение Пуассона. Используя определение распределения Пуассона и вышеизложенные рассуждения получаем

$$\eta = \min\{i \mid \prod_{j=1}^{i} \xi_i < e^{-\lambda}\} \tag{2.17}$$

2.3 Выборки индексов

Имеем задачу получения случайной выборки объема m значений из множества $\{1, \ldots, n\}$. В случае выборки с возвращением (возможны повторы) просто пользуемся примером 2.4. Для выборки без возвращения наиболее эффективен в программной реализации следующий алгоритм.

Универсальный алгоритм для выборки без повторения.

Входные данные: `n` – число индексов; `m` – длина выборки.

Выходные данные: `L[1..m]` – результирующий вектор индексов.

```
1. for i:=1 to n do
2.   I[i]:=i;
3. endfor;
4. for s:=n downto n-m+1 do
5.   l:=random(1,s);
6.   L[n-s+1]:=I[l];
7.   I[l]:=I[s];
8.   DEC(s);
9. endfor.
```

Замечание: В случае `n=m` мы получаем алгоритм случайных перестановок.

В случае не равных вероятностей индексов (i выбирается с вероятностью p_i) алгоритм изменяется следующим образом:

- кроме вектора I имеется вектор вероятностей $P = \{p_i\}$, этот вектор изменяется после каждого выбора;
- в строке 6 случайный индекс получается по формуле $\mathrm{l}=\min\{i \mid \xi < \sum_{j=1}^{i} p_j\}$, где ξ равномерно распределена на (0,1) (процедура `rand()`);
- массив P также изменяется:

```
8a. I[l]:=I[s]; V:=P[l]; P[l]:=P[s];
8b. for i:=1 to n-s do
8c.   P[i]:=P[i]/(1.0-V);
8d. endfor.
```

2.4 Моделирование псевдослучайных процессов с заданными маргинальным распределением и АКФ

В большинстве универсальных систем дискретного имитационного моделирования присутствуют только датчики независимых, одинаково распределенных случайных величин. Включение в системы моделирования датчиков некоторых основных случайных процессов не исправляет ситуацию, поскольку они не отображают все многообразие случайных процессов в реальных системах. Это требует включения в универсальные системы моделирования средств которые, с одной стороны, более удовлетворяли бы потребностям адекватного моделирования, а с другой – были бы достаточно просты в использовании и занимали не слишком большой объем. Одним из таких средств являются датчики псевдослучайных процессов с заданными маргинальным распределением и автокорреляционной функцией (АКФ). Такие процессы во многих случаях позволяют вполне адекватно описывать воздействия внешней среды на моделируемую систему. Пример важности учёта автокорреляции приведен на рис. 3.1, на котором представлены графики двух процессов с одним и тем же маргинальным экспоненциальным распределением с параметром $\lambda = 1$, но разными автокорреляционными функциями. Имитационная модель простой очереди M/M/1 с интенсивностью обслуживания $\mu = 1.5$ в случае независимых интервалов между поступлениями требований (6000 наблюдений) дало среднее время ожидания $\bar{w} = 7.94$. При использовании интервалов, заданных приведенными на рис. 3.1 процессами, были получены $\bar{w} = 12.136$ и $\bar{w} = 7.62$, соответственно. Разница с первым результатом значима при использовании t-критерия с 95% уровнем значимости.

[h!tb](110,75)(0.00,0.00)procs$_s$.*pcx*

Рассмотрим приближение одномерных процессов с заданными маргинальным распределением и автокорреляционной функцией рандомизированными цепями Маркова. Предложенный генератор весьма прост и компактен. Мы рассмотрим две его модификации: первоначальную, предложенную С.И. Молчаном [9, 4] и улучшенную автором в [5].

2.4.1 Общая схема алгоритма

Рандомизированная цепь Маркова (РЦМ) – это случайный процесс, основанный на цепи Маркова с числом состояний N и матрицей переходных вероятностей P, такой, что каждому состоянию i ставится в соответствие реализация случайной величины, распределенной согласно некоторой кумулятивной функции распределения $F_i(x)$.

Пусть $\xi_1, \xi_2, \ldots, \xi_n$ – множество независимых, одинаково распределенных случайных величин с непрерывной функцией распределения $F(x)$, и пусть $\xi_{(1)}, \xi_{(2)}, \ldots, \xi_{(n)}$ – эти же случайные величины,

упорядоченные в неубывающем порядке. Предполагается, что для всех значений времени правило упорядочения одинаково. Таким образом,

$$P(\xi_{(i)}(t) < x) = P(\xi_{(i)} < x) = F_i(x). \tag{2.18}$$

Очевидно, что

$$\sum_{i=1}^{N} F_i(x) = N \times F(x). \tag{2.19}$$

Обозначим через Ψ независимую от $\xi_{(i)}(t)$ однородную марковскую цепь с множеством состояний $i = 1, 2, ..., N$, вектором начальных вероятностей π_0 и двустохастической матрицей переходных вероятностей:

$$\begin{array}{l} (P(\Psi_{t+1} = j \mid \Psi_t = i)) = (p_{ij}) = P, \\ \sum\limits_{i=1}^{N} p_{ij} = 1; \quad \sum\limits_{j=1}^{N} p_{ij} = 1; \quad i, j = 1, 2, \ldots, N. \end{array} \tag{2.20}$$

Проведем рандомизацию этой цепи по правилу

$$\xi_t = \xi_{\Psi_t}(t), t = 0, 1, 2, \ldots \tag{2.21}$$

Далее приводятся основные свойства процесса, порождаемого этой РЦМ.

Конечномерное распределение процесса при условии

$$\pi_0 = (1/N, \ldots, 1/N) \tag{2.22}$$

определяется выражением:

$$\begin{aligned} P(\xi_{t_1} < x_1, \ldots, \xi_{t_m} < x_m) \quad = \quad & 1/N \times \sum_{j_1, \ldots, j_m = 1}^{N} [F_{j_1}(x_1) \times \ldots \times F_{j_m}(x_m) \times \\ & p_{j_1 j_2}(t_2 - t_1) \times \ldots \times p_{j_{m-1} j_m}(t_m - t_{m-1})]. \end{aligned}$$

Отсюда получается выражение для маргинального закона и автокорреляционной функции моделируемого процесса:

$$F_\xi(x) \quad = \quad 1/N \times \sum_{i=1}^{N} F_i(x) = F(x), \tag{2.2}$$

$$r'(\tau) \quad = \quad \frac{1}{N D_\xi} \sum_{i,j=1}^{N} M_i M_j \left(p'_{ij}(\tau) - \frac{1}{N} \right), \qquad \tau = 1, 2, \ldots, \tag{2.3}$$

где D_ξ – дисперсия маргинального распределения, $p'_{ij}(\tau)$ – ij-ый элемент матрицы переходных вероятностей, возведенной в степень τ, а

$$M_i = \int_{-\infty}^{\infty} x \, dF_i(x) \tag{2.4}$$

– математическое ожидание порядковой статистики $\xi_{(i)}$.

Для получения $dF_i(x)$ в этом выражении можно воспользоваться известной формулой для элемента вероятности порядковой статистики из [**?**]:

$$\begin{aligned} dF_i(x) &= \frac{\Gamma(n+1)}{\Gamma(i)\Gamma(N-i+1)} F^{i-1}(x)\left[1-F(x)\right]^{N-i} f(x)\,dx \\ &= \frac{N!}{(i-1)!(N-i)!} F^{i-1}(x)\left[1-F(x)\right]^{N-i} f(x)\,dx \\ &= (N-i+1)\begin{pmatrix} N \\ i-1 \end{pmatrix} F^{i-1}(x)\left[1-F(x)\right]^{N-i} f(x)\,dx. \end{aligned} \tag{2.5}$$

Если мы имеем полученную каким-либо образом матрицу переходных вероятностей, то алгоритм генерации РЦМ состоит из двух частей: получения нового состояния цепи Маркова и генерации случайного числа по соответствующему этому состоянию распределению вероятностей. Таким образом, алгоритм Молчана (G1) выглядит следующим образом:

Алгоритм G1.
Входные параметры:

N – число состояний Марковской цепи;

P – матрица переходных вероятностей;

n – требуемое число наблюдений процесса.

Выходные параметры:

$X(1 \times n)$ – вектор реализации процесса.

Шаг 1 (Предварительный). Поместить выборку из некоррелированной последовательности с заданным маргинальным распределением $F(x)$ в урну без возвращения. Счетчик k устанавливается в единицу. Начальное состояние цепи Маркова выбирается равновероятно из $1, 2, \ldots, N$.

Шаг 2. Из урны выбираются случайным образом $\xi_i,\ i = 1, \ldots, N$, и затем упорядочиваются неубывающим образом. Обозначим элементы упорядоченной выборки как $\xi_{(i)},\ i = 1, \ldots, N$.

Шаг 3. Согласно данной двустохастической матрицы переходных вероятностей определяется следующее (обозначим j) состояние цепи Маркова.

Шаг 4. ξ_j передается на выход как X_k, счетчик k увеличивается на 1.

Шаг 5. Если k превысило n, то завершение работы алгоритма, иначе на шаг 2.

Очевидны недостатки этого алгоритма:

- необходимо хранить выборку из N элементов;
- для получения каждого нового значения процесса необходимо сортировать выборку длиной N;
- в общем случае достаточно сложно получить аналитически выражение для M_i, знание которых необходимо для получения матрицы P (задача рассматривается ниже).

В основу улучшенной модификации алгоритма легло использование разделения области изменения процесса квантилями и замена порядковых статистик реализациями случайных величин из соответствующих усеченных распределений.

Алгоритм G2.
Входные и выходные величины те же, что и для G1.

Шаг 1 (Предварительный). Начальное состояние цепи Маркова выбирается равновероятно из $1, 2, \ldots, N$. Счетчик k устанавливается в единицу.

Шаг 2. Согласно данной двустохастической матрицы переходных вероятностей определяется следующее (обозначим j) состояние цепи Маркова.

Шаг 3. Генерируется псевдослучайное число ξ согласно усеченному на j-м межквантильном интервале распределению $F_j(x)$.

Шаг 4. ξ_j передается на выход как X_k, счетчик k увеличивается на 1.

Шаг 5. Если k превысило n, то завершение работы алгоритма, иначе на шаг 2.

Возможность замены выбора из упорядоченной последовательности на генерацию числа, распределенного по усеченному распределению обосновывается тем, что выражения (2.2) и (2.3) при этом сохраняют свою истинность.

Функция усеченного на (x_{i-1}, x_i) распределения $F_i(x)$ определяется выражением

$$F_i(x) = \begin{cases} 0, & \text{для} \quad x < x_{i-1}, \\ \frac{F(x)-F(x_{i-1})}{F(x_i)-F(x_{i-1})}, & \text{для} \quad x_{i-1} \le x < x_i, \\ 1, & \text{для} \quad x \ge x_i. \end{cases}$$

Если x_i представляет собой i-й квантиль маргинального распределения, выражение принимает более простой вид

$$F_i(x) = \begin{cases} 0, & \text{для} \quad x < x_{i-1}, \\ N(F(x) - \frac{i-1}{N}), & \text{для} \quad x_{i-1} \le x < x_i, \\ 1, & \text{для} \quad x \ge x_i. \end{cases} \tag{2.6}$$

Как отмечено выше, выражения (2.2) и (2.3) все еще истинны, но сейчас математические ожидания M_i нужно получать, используя распределения (2.6):

$$M_i = \int_{x_{i-1}}^{x_i} x\, dF_i(x) = x\, F_i(x) \Big|_{x_{i-1}}^{x_i} - \int_{x_{i-1}}^{x_i} F_i(x)\, dx = x_i - \int_{x_{i-1}}^{x_i} F_i(x)\, dx. \tag{2.7}$$

Для генерации случайных чисел, распределенных согласно $F_i(x)$, можно использовать известные точные и приближенные методы. Гораздо сложнее задача получения двустохастической матрицы переходных вероятностей, которая рассматривается в следующем пункте.

Таким образом, задача моделирования случайного процесса с требуемыми маргинальным распределением и АКФ сводится к решению системы (2.3) нелинейных уравнений степени τ. Такие решения найдены лишь для специального вида стохастической матрицы [9], тогда как для реальных задач необходимо иметь решение в общем случае, что возможно только сделать только численно.

2.4.2 Определение матрицы переходных вероятностей

Для определения двустохастической матрицы переходных вероятностей (МПВ) нам необходимо минимизировать расстояние между требуемой автокорреляционной функцией и АКФ моделируемого предложенным алгоритмом процесса по элементам МПВ. Если использовать обычную меру расстояния – сумму квадратов разностей в точках $\tau = 1, 2, \ldots$, то мы получаем следующую оптимизационную задачу:

$$\begin{aligned} \Phi(\tau) &= \sum_{\tau} (r(\tau) - r'(\tau))^2 \to min, \\ r'(\tau) &= \frac{1}{ND_\xi} \sum_{i,j=1}^{N} M_i M_j (P'_{ij}(\tau) - \tfrac{1}{N}), \end{aligned} \tag{2.8}$$

$$\sum_{i=1}^{N} P'_{ij} = 1; \qquad \sum_{j=1}^{N} P'_{ij} = 1; \quad i,j = 1,2,\ldots,N, \tag{2.9}$$

$$\forall i,j \quad 0 \le P'_{ij} \le 1, \tag{2.10}$$

где $P'_{ij}(\tau)$ суть элементы требуемой матрицы в степени τ, а $r'(\tau)$ – АКФ рандомизированной цепи Маркова, соответствующей этой МПВ и заданному маргинальному распределению.

Очевидно, что как вид целевой функции и ограничений, так и количество переменных (квадрат от числа состояний цепи Маркова) делают задачу оптимизации нетривиальной. Ниже рассмотрены различные подходы к ее решению.

Ранее использованные подходы

В [?] приведено утверждение автора алгоритма G1 С.И. Молчана о том, что вид задачи (3.2.7)-(3.2.9) не позволяет применять для ее решения градиентные методы. Вместо этого им был предложен эвристический метод стохастической оптимизации (алгоритм SO1), в котором, если это уменьшало значение целевой функции Ф, производились следующие изменения в матрице P' для случайно выбранной пары индексов:

$$\begin{aligned} P'_{ij}(m+1) &= P'_{ij}(m) \pm \delta, \\ P'_{kl}(m+1) &= P'_{kl}(m) \pm \delta, \\ P'_{il}(m+1) &= P'_{il}(m) \mp \delta, \\ P'_{kj}(m+1) &= P'_{kj}(m) \mp \delta. \end{aligned}$$

Процесс оптимизации заканчивался в случае, если за заданное количество шагов выбора случайных пар не наблюдалось уменьшения значения функционала. Очевидно, что этот алгоритм сохраняет двустохастичность МПВ P' и результат всегда допустим, если пробный шаг δ производится без нарушения ограничения $\forall i,j \; P'_{ij} \in [0,1]$. Однако еще более очевидно что этот алгоритм крайне неэффективен. Для достижения значения разницы меньше 10% в случае $N = 10$ и $\tau_{\max} = 20$ требовалось порядка 20,000-27,000 проб в зависимости от АКФ и математических ожиданий распределений состояния. Как показатель разницы при аппроксимации здесь используется

$$\varepsilon = \sqrt{\sum_{\tau} \frac{(r(\tau) - r'(\tau))^2}{\tau_{\max}}} \cdot 100\%. \tag{2.11}$$

Автором (в той же статье) модификация этого алгоритма (алгоритм SO2). В SO2 рабочим шагом является

$$\begin{aligned} P'_{ij}(m+1) &= P'_{ij}(m) \pm s, \\ P'_{kl}(m+1) &= P'_{kl}(m) \pm s, \\ P'_{il}(m+1) &= P'_{il}(m) \mp s, \\ P'_{kj}(m+1) &= P'_{kj}(m) \mp s, \end{aligned} \tag{2.12}$$

где s определяется с помощью линейной аппроксимации по точкам $(0, \Phi(m))$ и $(\pm, \Phi_\delta(m))$, где $\Phi_\delta(m)$ есть улучшенное значение функционала на пробном шаге (решение линейного уравнения). Исходя из ограничений двустохастичности имеем:

$$s = \max\left\{\delta, \min\left\{\rho, \delta\frac{\Phi(m)}{\Phi(m) - \Phi_\delta(m)}\right\}\right\}, \tag{2.13}$$

где ρ, в свою очередь, определяется как

$$\min\{P'_{ij}(m), P'_{kl}(m), 1 - P'_{il}(m), 1 - P'_{kj}(m)\} \tag{2.14}$$

или

$$\min\{P'_{il}(m), P'_{kj}(m), 1 - P'_{ij}(m), 1 - P'_{kl}(m)\}, \tag{2.15}$$

в зависимости от уменьшаемой пары. Если изменение на s в P' в выражении (3.2.10) не уменьшает функционал, используется шаг δ. В то время, когда этот метод был предложен, одномерная оптимизация по s в выбранном случайном направлении была невозможна по причине малой производительности использовавшейся ЭВМ (СМ 1420). Хотя эта модификация сократила число проб в среднем в три раза, алгоритм SO2 также очень неэффективен.

Многочисленные эксперименты показали, что SO2 в среднем требует около 40 минут машинного времени на PC с процессором Intel Pentium III 600MHz для получения 16x16 МПВ с 3% разницей в АКФ. Кроме того, он не гарантирует сходимость. Поэтому были сделаны попытки ускорить решение задачи и, в частности, опровергнуть утверждение Молчана о невозможности применения градиентных методов оптимизации. Этому способствовало также появление мощных ПЭВМ с высокой скоростью вычислений и большим объемом памяти.

Были проверены методы: скользящего допуска, проекции градиентов Розена и метод наискорейшего спуска с использованием штрафных функций. Эксперименты показали, что последний метод дает наилучший результат.

Использование метода Ньютона со штрафными функциями

Прежде всего отметим, что возможно достаточно просто избавиться от ограничений на двустохастичность, уменьшив при этом размерность задачи. Это делается очевидными подстановками:

$$\begin{aligned} P'_{iN} &= 1 - \sum_{j=1}^{N-1} P'_{ij}, \quad P'_{Ni} = 1 - \sum_{j=1}^{N-1} P'_{ji}, \quad i = 1, \ldots, N-1, \\ P'_{NN} &= \sum_{i,j=1}^{N-1} P'_{ij} - N + 2. \end{aligned} \tag{2.16}$$

После этого возможно превратить нашу задачу в задачу безусловной оптимизации добавлением штрафной функции:

$$\begin{aligned} &Q(P'^{(k)}_{ij}, \rho^{(l)}_{ij}, \delta^{(m)}_{ij}) = \\ &\Phi(P'^{(k)}_{ij}) + \sum_{i,j=1}^{N-1} \left[\rho^{(l)}_{ij} U^{(k)}_{ij} (P'^{(k)}_{ij})^2 + \delta^{(l)}_{ij} V^{(k)}_{ij} (P'^{(k)}_{ij} - 1)^2 \right], \\ &i, j = 1, \ldots, N-1, \end{aligned}$$

где

$U^{(k)}_{ij}$ = { 1 P'$_{ij} < 0$,
0 иначе.(), ={ 1 P'$_{ij} > 1$,
0 иначе.(),

(весовые коэффициенты), где l есть количество итераций с момента $P'^{(k)}_{ij} < 0$, $l \le 100$;

(весовые коэффициенты), где m есть количество итераций с момента $P'^{(k)}_{ij} > 1$, $l \le 100$.

Производные dQ/dp_{uv} $1 \le u, v \le N$ определяются следующим выражением (далее (k) и апостроф у P опущены для лучшей читаемости выражений):

$$\begin{aligned} \frac{dQ}{dp_{uv}} &= \frac{d\Phi}{dp_{uv}} + 2 \sum_{i,j=1}^{N-1} \left[\rho^{(l)}_{ij} U_{ij} P_{ij} - \delta^{(m)}_{ij} V_{ij} (P_{ij} - 1) \right], \\ \frac{d\Phi}{dp_{uv}} &= \frac{1}{ND_\xi} \sum_{\tau} \left[\frac{1}{ND_\xi} \sum_{i,j=1}^{N} M_i M_j \left(P_{ij}(\tau) - \frac{1}{N} \right) - r(\tau) \right] \times \end{aligned}$$

$$\sum_{i,j=1}^{N} M_i M_j \frac{dP_{ij}(\tau)}{dp_{uv}}.$$

При получении градиента в нашем случае его более удобно иметь в виде матрицы производных.

$$\frac{dP^\tau}{dp_{uv}} = \left(\frac{dP_{ij}(\tau)}{dp_{uv}}\right). \tag{2.17}$$

Очевидно,

$$\frac{dP^\tau}{dp_{uv}} = P \cdot \frac{dP^{\tau-1}}{dp_{uv}} + \frac{dP}{dp_{uv}} \cdot P^{\tau-1}. \tag{2.18}$$

Для уменьшения числа операций матрицы P^τ, полученные на каждой итерации при вычислении функции, сохраняются для вычисления производной. Тем самым, ценой затрат памяти, существенно сокращается время вычислений. Кроме того можно сократить объем вычислений учитывая особенность матрицы $D = dP/dp_{uv}$: она имеет только четыре ненулевых элемента (благодаря подстановкам выражений (3.2.11) в (3.2.12)):

$$d_{uv} = d_{NN} = 1,\ d_{uN} = d_{Nv} = -1. \tag{2.19}$$

Благодаря этому мы имеем

$$\forall i \neq u, N \forall j \left(\frac{dQ}{dp_{uv}} \cdot P^{\tau-1}\right)_{ij} = 0, \tag{2.20}$$

$$\forall j \left(\frac{dQ}{dp_{uv}} \cdot P^{\tau-1}\right)_{uj} = P_{uj}(\tau-1) - P_{Nj}(\tau-1), \tag{2.21}$$

$$\forall j \left(\frac{dQ}{dp_{uv}} \cdot P^{\tau-1}\right)_{Nj} = -P_{uj}(\tau-1) + P_{Nj}(\tau-1). \tag{2.22}$$

Обозначим $P_{uj}(\tau-1) - P_{Nj}(\tau-1)$ как $R_{uj}(\tau-1)$.
Тогда

$$\left(\frac{dP^\tau}{dp_{uv}}\right)_{uj} = \left(P \cdot \frac{dP^{\tau-1}}{dp_{ij}}\right)_{uj} + R_{uj}(\tau-1), \tag{2.23}$$

$$\left(\frac{dP^\tau}{dp_{uv}}\right)_{Nj} = \left(P \cdot \frac{dP^{\tau-1}}{dp_{ij}}\right)_{Nj} - R_{uj}(\tau-1). \tag{2.24}$$

Этот подход позволяет рассчитать (3.2.12), используя только одну операцию перемножения матриц.

Эксперименты показали очень быстрое (3-5 шагов) достижение области минимума с разницей АКФ 8–10% и последующее резкое замедление сходимости. Это замедление существенно зависит от рассматриваемой длины АКФ: для $n < 10$ оно почти неощутимо, тогда как для $n > 30$ достижение оптимума замедляется очень сильно. Очевидно, что этот эффект наблюдается благодаря потере значимости при вычислениях с большим количеством сумм произведений многих малых чисел. Это количество вычислений растет с τ экспоненциально. Кроме того, оптимизируемая функция имеет действительно очень медленное уменьшение в области минимума. Использование специальных численных методов и высокоточных вычислений должно исправить ситуацию.

В случае необходимости получения высокой точности аппроксимации предлагается следующая схема оптимизации.

Шаг 1. Ввод начальной двустохастической матрицы, например, трехдиагональной или с равными элементами.

Шаг 2. Нахождение градиентным методом матрицы для случая когда длина АКФ ограничивается малой величиной (порядка 10-15).

Шаг 3. С использованием результата предыдущего шага найти МПВ для случая АКФ реально необходимой длины.

Шаг 4. Если точность недостаточна, использовать для уточнения случайный поиск с оптимизацией по направлению.

Для экспериментов было выбрано два примера: один из худших по точности приближения примеров из [?] и приближение процесса интервалов между последовательными ошибками в передаче сообщений из [?].

2.4.3 Пример с регулярной, быстро сходящейся к нулю АКФ

В [?] приведена таблица, показывающая качество приближения и затраченное на него машинное время для различных маргинальных распределений и АКФ. Один из наихудших по качеству приближений пример характеризуется следующим:

- Маргинальное распределение – экспоненциальное с параметром $\lambda = 1$;
- число состояний цепи Маркова $N = 10$;
- АКФ $r(\tau) = e^{-0.1\tau}\cos(0.4\tau)$, с $\tau_{\max} = 40$.

Ниже проведено сравнение алгоритмов G1 и G2 для этого примера.
Вычисление математических ожиданий
Алгоритм G1. Согласно выражению (3.2.4) плотность ξ_i в нашем случае составляет

$$f_i(x) = \lambda(N-i+1)\binom{N}{i-1}\left(1-e^{-\lambda x}\right)^{i-1} e^{-\lambda(N-i+1)x}, \tag{2.25}$$

и, соответственно, математическое ожидание ξ_i имеет вид

$$\begin{aligned} M_i &= \int_0^\infty x f_i(x)\,dx = \int_0^\infty x\lambda(N-i+1)\binom{N}{i-1}\times \\ &\left(1-e^{-\lambda x}\right)^{i-1} e^{-\lambda(N-i+1)x}\,dx = \\ &\frac{1}{\lambda}(N-i+1)\binom{N}{i-1}\int_0^\infty t\left(1-e^{-t}\right)^{i-1} e^{-(N-i+1)t}\,dt = \\ &\frac{1}{\lambda}(N-i+1)\binom{N}{i-1}\sum_{j=0}^{i-1}\frac{(-1)^{i-j-1}}{N-j}\binom{i-1}{j}\times \\ &\int_0^\infty (N-j)te^{-(N-j)t}\,dt. \end{aligned}$$

Последний интеграл представляет собой математическое ожидание экспоненциального распределения с параметром $N-j$, поэтому окончательно получаем для M_i:

$$M_i = \frac{N-i+1}{\lambda}\binom{N}{i-1}\sum_{j=0}^{i-1}\frac{(-1)^{i-j-1}}{(N-j)^2}\binom{i-1}{j} = \frac{K_i}{\lambda}, \tag{2.26}$$

где K_i представляют собой значения, зависящие только от i и N. N фиксировано как некоторое не очень большое значение, не зависящее от параметра распределения λ. Когда N зафиксировано, все K_i могут быть рассчитаны без затруднений.

Алгоритм G2. Кумулятивная функция распределения $F_i(x)$ определяется выражением (3.2.5), а математические ожидания M_i – выражением (3.2.6). В случае экспоненциального маргинального распределения получаем, что i-й квантиль равен

$$x_i = -\frac{1}{\lambda}\ln\left(\frac{N-i}{N}\right),\ 1 \le i \le N-1, \tag{2.27}$$

где N есть число интервалов (для общности выражений добавим крайние значения: $x_0 = 0$ и $x_N = \infty$). Выражение для кумулятивной функции усеченного на межквантильном интервале экспоненциального распределения получим, воспользовавшись (3.2.5) и (3.2.18). В пределах i-го интервала получаем:

$$F_i(x) = N(1 - e^{-\lambda x}) - i + 1. \tag{2.28}$$

Легко получаем соответствующие математические ожидания

$$\begin{aligned} M_i &= \tfrac{1}{\lambda}\left(1 + \ln\tfrac{N(N-i)^{N-i}}{(N-i+1)^{N-i+1}}\right) = \\ &\quad \tfrac{1}{\lambda}\left(1 + \ln N + (N-i)\ln(N-i) - \right. \\ &\quad \left.(N-i+1)\ln(N-i+1)\right) = \tfrac{K_i^*}{\lambda}, \\ &\quad i = 1, \ldots, N-1. \\ M_N &= \tfrac{1}{\lambda}(1 + \ln N) = \tfrac{K_N^*}{\lambda}. \end{aligned} \tag{2.29}$$

Как и в предыдущем случае, M_i обратно пропорциональны параметру распределения λ. Отметим, что вычисление K_i^* для алгоритма G2 намного проще, чем вычисление K_i для алгоритма G1, так как оно не требует сумм и биномиальных коэффициентов.

Определение матрицы переходных вероятностей

Подход к получению МПВ один и тот же для обоих алгоритмов, но результаты, конечно, различны. Интересно, однако, что для приведенного выше примера, а также для не рассматриваемого здесь примера равномерного распределения, для всех рассматриваемых АКФ и числа состояний цепи Маркова, минимальное значение разницы АКФ всегда получалось меньше для случая алгоритма G2 (в среднем на 20%).

[h!tb](96,120)(0.00,0.00)acfs$_s$.*pcx*2.4.3

В [4] для этого примера, для случая алгоритма G1, разница, достигнутая с помощью алгоритма SO2 с 25000 случайными шагами, составила около 10%. Использование предложенной выше схемы совместно с применением алгоритма G2 позволило достигнуть гораздо лучшей точности. Вначале, при последовательном выполнении градиентной оптимизации с длиной АКФ от 5 до 40 с шагом 5 была получена разность 9.6%. Последующее выполнение 500 шагов случайного поиска (3.2.10) с одномерной оптимизацией дало разницу в 6.1%. Время вычислений составило около 3 минут на PC с упомянутой выше конфигурацией. Разница в 2.4% была достигнута для этих АКФ и маргинального распределения при использовании цепи Маркова с 16 состояниями. В этом случае та же вычислительная схема потребовала более 20 минут процессорного времени, а разница АКФ после градиентной оптимизации составила 14.6%. Число шагов случайного поиска было увеличено до 2500.

[h!tb](96,127)(0.00,0.00)histcox.pcx Гистограмма распределения интервалов между ошибками

Шаг генерации

Для собственно процесса генерации нам необходимо иметь датчик экспоненциально распределенных случайных величин для алгоритма G1 и датчик усеченного экспоненциального распределения для алгоритма G2. В первом случае используется хорошо известная получаемая методом обратной функции формула $\eta = -\ln\xi/\lambda$. Во втором, используя тот же метод и формулу (3.2.19), легко получаем

$$\eta_i = -\ln[\xi(e^{-\lambda x_i} - e^{-\lambda x_{i-1}}) + e^{-\lambda x_{i-1}}]. \tag{2.30}$$

Поскольку x_i в нашем случае есть квантили, подставляем (3.2.18) и получаем

$$\begin{aligned} \eta_i &= -\ln\left(\frac{N-i+1}{N} - \frac{\xi}{N}\right), \\ \eta_N &= \ln N - \ln \xi/\lambda. \end{aligned}$$

Поскольку алгоритм G2 не требует шага сортировки, он очевидно быстрее, чем G1. На генерацию 100,000 чисел с использованием цепи Маркова с 16 состояниями, в случае алгоритма G2 было потрачено 1.86 с, а в случае алгоритма G1 – 25.04 с на упомянутом выше компьютере.

Точность аппроксимации АКФ проиллюстрирована на рис. 3.2.

2.4.4 Пример дискретного маргинального распределения и нерегулярной АКФ

Рассмотрим пример приближения процесса поступления последовательных ошибок в передаче сообщений из [?] Этот пример очень интересен, но труден для решения.

Описание примера

Пример взят из [?]. Выборка состоит из 672 значений интервалов между ошибками в передаче данных. Пример нуждается в анализе распределения и вычислении АКФ. Гистограмма выборки представлена в рис. 3.3.a. Из-за нескольких очень больших значений (длинный хвост), эта гистограмма не достаточно информативна. На рис. 3.3.b, представлена гистограмма этой же выборки без 20 самых больших значений. Ни одно из "классических" распределений не удовлетворяет этой гистограмме из-за очень длинного хвоста распределения. Распределение Парето дает наилучшее приближение, но критерий χ^2 (10% порог чувствительности) не удовлетворен. Таким образом, выборочное распределение используется в процессе генерации для минимизации ошибок. АКФ, полученная выборкой, имеет довольно нерегулярную форму, и это будет обсуждено позже. Число использованных состояний, N, было равно 16 и 21.

Вычисление математических ожиданий

В случае алгоритма G1 мы получили математические ожидания методом Монте-Карло, используя 1000 выборок N случайно выбранных чисел из целой выборки. В случае алгоритма G2 мы получили математические ожидания простой обработкой данных соответствующих частей упорядоченной основной выборки. Очевидно, это потребовало меньше усилий, чем в случае алгоритма G1.

Выбор числа состояний и длины АКФ

Отметим, что длина нашей выборки – $672 = 16 \times 42 = 21 \times 32$. Следовательно, N=16 или 21 был выбран естественным образом. Что касается длины АКФ, выбор не настолько очевиден. Рассмотрение АКФ показывает только два значимых значения из первых 15 ($\tau = 2, 13$), проверка 25 значений не добавляет новых значимых значений. Сначала, были проведены эксперименты с $n = 25$, и результаты не были столь удовлетворительными, как ожидалось. Проверка еще 10 значений АКФ показывает еще одно существенное значение при $\tau = 32$. Новые вычисления и эксперимент дали определенно лучшие результаты (см. рис. 3.3). Увеличение длины АКФ до 35 дает более адекватный выход алгоритма. Однако время, потраченное для получения матрицы P, также увеличилось значительно, как показано в следующем пункте.

[h!tb](96,61)(0.00,0.00)acfs2.pcx Приближение АКФ для распределения интервалов между ошибками

Матрица переходных вероятностей

Чтобы получить матрицу переходных вероятностей, используемых алгоритмами G1 и G2, мы нуждаемся в намного больших усилиях, чем в случае предыдущего примера. Возможно, это происходит из-за равенства нескольких первых математических ожиданий одному и тому же значению (а

именно, два первых математических ожидания алгоритма G1 и четыре для Алгоритма G2 оказались равны 1) и очень большой разности между самыми маленькими и самыми большими математическими ожиданиями (в случае $N = 21$ M_{21}=21314.1 или 24848.5 и M_{20}=4131.7 или 4225.5 для алгоритмов G1 и G2, соответственно). Оптимизация градиентным методом дает приблизительно 30%-ю разность, и последующий случайный поиск дает приблизительно 3.16 (3.27)%-ю при $N = 16$ и приблизительно 2.01(2.03)%-ю разность при $N = 21$ для алгоритма G1(G2). К сожалению, мы должны признать, что потребовалось приблизительно 20 и 32 часа счета соответственно для Алгоритмов G1 и G2, чтобы достичь этой точности. На рис. 3.4, ясно показано, что точность аппроксимации уменьшается с ростом индекса АКФ. Таким образом, предложенная схема оптимизации недостаточно эффективна в случае дискретного распределения с несколькими равными математическими ожиданиями и очень длинным хвостом. Для этих условий мы должны построить другие схемы оптимизации. Обратим внимание на то, что Алгоритм G1 дает слегка лучшую аппроксимацию. Однако напомним, что задача оптимизации должна быть решена только однажды для произвольного числа порождений процесса.

Шаг генерации

Когда мы имеем матрицу переходных вероятностей, шаг генерации прост для обоих алгоритмы. Для Алгоритма G1, предварительные выборки N значений получены из целого выборочного распределения формулой

$$x_i = X^*_{[m\xi]+1}, \tag{2.31}$$

где m - размер выборки X (672 в нашем случае), X^* - выборка, упорядоченная в неубывающем порядке, и ξ - случайное число, равномерно распределенное на [0,1). Для Алгоритма G2, уникальное значение выбирается тем же самым способом из i-ой части X^*, где i - текущее состояние цепи Маркова. Следовательно, преимущество Алгоритма G2 по отношению к G1 – отсутствие предварительного осуществления выборки и шага упорядочения. Для генерации 100,000 чисел для рандомизированной цепи Маркова с 21 состоянием, 2.04 секунда была потрачена в случае Алгоритма G2 и 27.16 секунды в случае Алгоритма G1, соответственно. Обратим внимание, что предложенный алгоритм снова быстрее чем предыдущий. Небольшая уступка в точности в этом примере полностью компенсируется скоростью.

2.4.5 Влияние выбранной длины АКФ

[h!tb](97,63)(0.00,0.00)acf$_i$*l.pcx*($G2$).

На Рисунке 2.4.5 проиллюстрирована, на искусственном примере АКФ

$$r(\tau) = \{\, 1 - 0.05\tau$$

$0<\tau \leq 20$,
0 иначе,(2.32)

важность правильного усечения длины АКФ n для шага получения МПВ. При $n = 20$, АКФ хорошо аппроксимирована на $[1, \ldots, 20]$ а затем далеко отходит от нуля. Для $n = 25$, получается лучший результат, а при $n = 30$ получаем действительно хорошую аппроксимацию. При этом на интервале $[1, \ldots, 20]$ имеем хорошую аппроксимацию для всех этих случаев. При моделировании число состояний цепи Маркова N было также равно 16.

2.5 Случайные строки

Пусть $X_n(p) = x_1x_2...x_n$ - случайная строка бит длины n, такая, что $P(x_i = 1) = p$, $P(x_i = 0) = q$ и x_i независимы в совокупности.

Простейший способ получения реализации такой случайной строки заключается в ее поразрядной генерации, что требует обращение к базовому датчику для каждого разряда, что крайне неэффективно для длинных строк. Предлагается следующий приближенный метод, точный для вероятности $p = l/2^i,\ 1 \le l \le 2^i - 1$.

Рассмотрим класс логических функций вида $\mathrm{L}_N(X_1, X_2, ..., X_N) = ((...(X_1\ S_1\ X_2)\ S_2\ X_3)\ S_3\ X_4\ ...)S_{N-1}$ есть функция $\vee$ либо $\wedge$, X_i - строки бит.

Для этого класса функций справедлива следующая теорема. Пусть X_i есть реализации $X_n(1/2)$. Тогда существует, и притом единственная, функция L_l^i из класса функций (8), такая, что вероятность появления единицы в разряде результирующей строки равна $l/2^i,\ 1 \le l \le 2^i - 1$. Применим метод математической индукции.

Обозначим $\mathcal{E}(X)$ вероятность появления 1 в произвольном разряде строки X. Для $X_n(p)\ \mathcal{E}(X_n(p)) = p$.

Шаг 1. Для $i = 1$ имеем * $\mathrm{L}_1^1 = X_1,\ \mathcal{E}(L_1^1) = \mathcal{E}(X_1) = \frac{1}{2}$;

Для $i = 2$ имеем * $\mathrm{L}_1^2 = X_1 \wedge X_2,\ \mathcal{E}(L_1^2) = \frac{1}{4}$;
$L_2^2 = L_1^1 = X_1,\ \mathcal{E}(L_2^2) = \frac{1}{2} = \frac{2}{4}$;
$L_3^2 = X_1 \vee X_2,\ \mathcal{E}(L_3^2) = \frac{3}{4}$.

Шаг 2. Пусть теорема верна для некоторого $i = s$. Покажем, что она верна и для $i = s + 1$.

Для $l = 2k,\ k = 1, ..., 2^{s-1}$ имеем $L_{2k}^{s+1} \equiv L_k^s$.

Рассмотрим строки $L_{M(k)}^{s+1}$ и $L_{N(k)}^{s+1}$, $k = 1, 2, \ldots, 2^s - 1$, получающиеся из L_{2k-1}^s с помощью конъюнкции либо дизъюнкции с X_{s+1} соответственно.

$\mathrm{E}(\mathrm{L}_{M(k)}^{s+1}) = \mathcal{E}(L_{2k-1}^s \wedge X_{s+1}) = \frac{1}{2}\mathcal{E}(L_{2k-1}^s) = \frac{2k-1}{2^{s+1}}$,

откуда $M(k) = 2k - 1$;

$$\begin{aligned}\mathcal{E}(L_{N(k)}^{s+1}) &= \mathcal{E}(L_{2k-1}^s \vee X_{s+1}) = \mathcal{E}(L_{2k-1}^s) + \frac{1}{2} - \frac{1}{2}\mathcal{E}(L_{2k-1}^s) = \\ &= \frac{2k-1}{2^s} + \frac{1}{2} - \frac{2k-1}{2^{s+1}} = \frac{2k + 2^s - 1}{2^{s+1}},\end{aligned}$$

откуда $N(k) = 2k + 2^s - 1$.

Очевидно, что для всех $k_1 \ne k_2$ выполняется $M(k_1) \ne N(k_2)$ и $N(k_1) \ne M(k_2)$.

Действительно, пусть существуют $k_1 \ne k_2$ такие, что $N(k_1) = M(k_2)$, тогда из (7) и (8) имеем * $2\mathrm{k}_1 + 2^s - 1 = 2k_2 - 1$
$k_2 = k_1 + 2^{s-1} 1 \le k_1,\ k_2 \le 2^{s-1}$ и таких k_1 и k_2 не существует.

Отметим, что $N(k)$ и $M(k)$ нечетны.

Таким образом, построено множество $\mathcal{L}_{s+1}$ функций вида (5): * $\mathrm{L}_{s+1} = \left\{L_{2k}^{s+1}\right\} \cup \left\{L_{N(k)}^{s+1}\right\} \cup \left\{L_{M(k)}^{s+1}\right\}$

При этом все элементы $\mathcal{L}_{s+1}$ различны и * $\|\mathcal{L}_{s+1}\| = 2^s - 1 + 2^{s-1} + 2^{s-1} = 2^{s+1} - 1$.

Из последнего факта и следует доказательство теоремы.

Согласно доказанной теореме правило построения L_l^i выражается следующим рекурсивным алгоритмом.

Шаг 1. Если $L_l^i = L_1^1$, то $L_l^i = X_1$.

Шаг 2. Если $\frac{l}{2^i} < \frac{1}{2}$, то S_{i-1} есть "$\wedge$"и $L_l^i = L_l^{i-1} \wedge X_i$.

Шаг 3. Если $\frac{l}{2^i} > \frac{1}{2}$, то S_{i-1} есть "$\vee$"и $L_l^i = L_l^{i-1} \vee X_i$.

Нетрудно заметить, что данное правило идентично построению двоичного кода для исходной вероятности $p = \frac{l}{2^i}$ ("∧"соответствует 0, а "∨ единице разложения, последняя единица разложения соответствует $L_1^1 = X_1$). Так, если $p = 19/32 = 0.10011$, то * L(X$_1, X_2, X_3, X_4, X_5$) = $(((X_1 \vee X_2) \wedge X_3) \wedge X_4) \vee X_5$.

Что касается исходных строк X_i, то они легко получаются конкатенацией двоичных представлений реализаций базового датчика ξ_i, хотя лучше при этом не использовать их последние биты.

2.6 Случайные графы

2.6.1 Основные обозначения, понятия и подходы

Обозначим произвольный неориентированный граф с N вершинами, M ребрами и без кратных ребер как $G(N, M)$. Для генерации случайных графов может быть использованы несколько подходов. Первый и наименее эффективный подход (но все еще широко используемый) - *метод случайных испытаний*. Случайный график с данным числом вершин (и, возможно, ребер) генерируется каким-либо образом и затем его свойства проверяются, на соответствие заданным ограничениям. В нашем случае это означает, по крайней мере, что граф должен быть связным. Очевидно, что число испытаний быстро растет с ростом числа вершин. Действительно, общее количество различных граф $G(N, M)$ (со всем возможным изменением нумерации вершин)

$$S = \left(\begin{array}{c} \frac{N(N-1)}{2} \\ M \end{array} \right) \frac{N(N-1)}{2}! \tag{2.33}$$

Рассмотрим пример генерации $G(N, M)$. Самый простой путь состоит в том, чтобы использовать матрицу смежности (VV). Пусть VV изначально состоит из нулевых элементов. В цикле от 1 до M пара случайных чисел i и j выбирается из $[1, \ldots, N]$, $\quad i \neq j$ и, если $VV_{ij} = 0$ то полагаем $VV_{ij} = 1$ и $VV_{ji} = 1$, иначе выбор повторяется. Этот самый простой алгоритм требует следующее среднее число испытаний:

$$N_c = 1 + n \left(\frac{1}{n-1} + \frac{1}{n-2} + \ldots + \frac{1}{n-M+1} \right), \tag{2.34}$$

где $n = N(N-1)/2$.

Обратим внимание на то, что выбор пары индексов методом испытаний сам требует в среднем $n/(n-1)$ попыток, таким образом общее число обращений к генератору случайных чисел в среднем равно

$$N_r = \frac{n}{n-1} \left[1 + n \left(\frac{1}{n-1} + \frac{1}{n-2} + \ldots + \frac{1}{n-M+1} \right) \right], \tag{2.35}$$

Таким образом, например, для $N = 10,\ M = 20,\ N_r = 25.67$. Следовательно, предпочтительнее использование выборки без повторения. Сначала создается упорядоченное множество пар $(i, j), \quad i = 1, \ldots, N-1, \quad j = i+1, \ldots, N$. Число пар – n. Затем индексы нужных пар выбираются с помощью алгоритма выборки без повторения.

Следующий базовый подход – “последовательный рост": каждый новый элемент добавляется к уже сгенерированному графу без нарушения ограничений и гарантируя его свойства. И снова имеются две возможности: использовать метод случайных испытаний или “умный"метод на каждом шаге.

В качестве “умного"метода мы будем использовать *метод допустимого выбора (МДВ)*, в котором на каждом шаге выбор делается только от множества элементов, который 1) сохраняет граф в данном классе, и 2) не позволяет нарушать ни одно из ограничений. Прежде, чем перейти к обсуждению примеров, введем обозначения:

X_i - множество вершин сгенерированного графа на i-ом шаге;

E_i - множество ребер сгенерированного графика на i-ом шаге;

A_i - множество ребер, которые возможно добавить к сгенерированному графу на i-ом шаге;

In_i - множество ребер, которые должны быть добавлены к A_i перед $i+1$-ым шагом;

Ex_i - множество ребер, которые должны быть исключены из A_i перед $i+1$-ым шагом;

v_i - i-я вершина графа;

e_{ij} - ребро, которое соединяет v_i и v_j;

$X(G)$ - множество вершин графа G;

$E(G)$ - множество ребер графа G;

$C(N)$ - полный N-вершинный граф.

Таким образом, $A_{i+1} = A_i \setminus Ex_i \cup In_i$. Некоторые ограничения могут приводить к пустому A_{i+1}. Если это произошло не из-за невозможности получения графа с данными свойствами вообще (это должно быть проверено перед генерацией), то производится откат на один шаг. Если e_{st} было ребром, выбранным последним перед тупиком, то оно переводится из A_i в Ex_i.

Пусть L есть множество ограничений, назначенных данному пространству графов и $R(G, L)$ есть индикаторная функция :

$$R(G,L) = \begin{cases} -1 & \text{если } G \text{ нарушает ограничения,} \\ 0 & \text{если } G \text{ находится на границе,} \\ 1 & \text{в остальных случаях.} \end{cases} \tag{2.36}$$

Тогда общее правило для МДВ:

1. если $R(G(X_i, E_i)) = -1$ и e_{st} – ребро, выбранное на последнем шаге, то взять шаг назад и удаление est из A_i.
2. если $R(G(X_i, E_i)) = 0$, то ребра, которые при добавлении к текущему графу могут нарушить ограничения, добавляются к Ex_i.
3. если $R(G(X_i, E_i)) = 0$ и e_{st} – ребро, выбранное на последнем шаге, то $Ex_i = e_{st}$.

Эффективность алгоритма генерации в сильной степени зависит от процедуры проверки нарушений и их предсказания. Если состояние на границе ограничения обнаруживается просто, то возможно не позволять любые нарушения вообще. Так образом, если степень вершины достигла ее верхнего предела, то достаточно более не рассматривать свободные ребра, связанные с этой вершиной. Но, например, чтобы проверить заранее нарушение максимально допустимого диаметра графа для всех ребер из A_i может потребовать большого количества действий, поэтому свойство проверяется после выбора ребра и, возможно, делается откат.

2.6.2 Генерация случайных деревьев

Простым примером служит генерация случайных деревьев в случае равной вероятности существования ребер. Если используется метод случайных испытаний, то после каждого нового выбора ребра, проверяется, не приводит ли добавление этого ребра к циклу или мульти-ребру. Если да то ребро отклоняется и повторяется выбор. Если же используется МДВ, то сначала любая вершина выбирается равновероятно в качестве корня, пусть это будет v_f. Тогда

$$A_1 = \{e_{f1}, e_{f1}, e_{f2}, \ldots, e_{f,f-1}, e_{f,q+1}, \ldots, e_{fN}\} \tag{2.37}$$

If on the i-th step the edge e_{st} is selected (t-th node is new) then

$$Ex_i = \{e_{ct} | v_c \in X_i\} \quad (2.38)$$
$$In_i = \{e_{td} | v_d \in \bar{X}_i\} \quad (2.39)$$

Опишем теперь этот алгоритм формально и обсудим затем более сложные постановки задачи генерации случайных деревьев с ограничениями.

Алгоритм генерации случайного N-вершинного дерева

Для генерации случайного N-вершинного дерева (изоморфизм возможен) предлагается следующий быстрый алгоритм (см. Рис. 1):

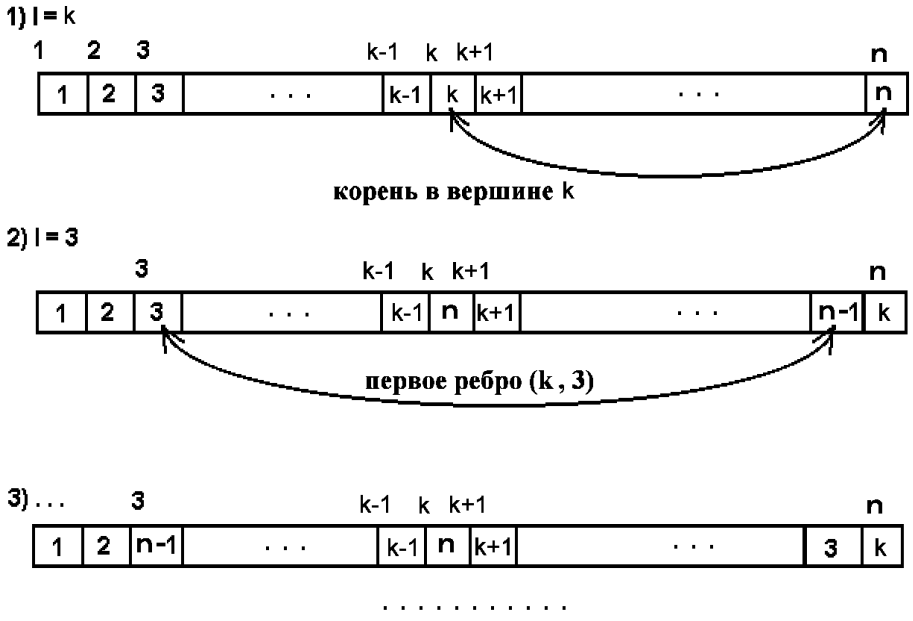

Рис. 2.1: Иллюстрация к быстрому алгоритму генерации случайного дерева

Быстрый алгоритм генерации случайного дерева

Входные данные: `N` - число вершин.
Выходные данные: `OU[1..N-1]` и `IO[1..N-1]` – векторы номеров начальных и конечных вершин для ребер результирующего дерева (i-е ребро соединяет вершины `OU[i]` и `IO[i]`).

```
1.  for i:=1 to N do
2.   I[i]:=i;        \\ Начальный вектор номеров вершин
3.  endfor;
4.  l:=random(1,N);   \\ Выбор корня
5.  I[N]:=I[l]; I[l]:=N; \\ Перестановка с последней вершиной
6.  for m:=N-1 downto 1 do
7.   t:=random(1,m);       \\ Новая вершина
8.   s:=random(m+1,N);     \\ Вершина "роста"
9.   OU[m]:=I[t]; IO[m]:=I[s]; \\ Регистрация ребра
10.  K:=I[t]; I[t]:=I[m]; \\ Перестановка: от N-m+1 до N -
     I[m]:=K;            \\ вершины, уже вошедшие в дерево
11. endfor.
```

Замечание: очевидно, что этот алгоритм требует только $2N-1$ обращений к генератору случайных чисел и использует минимальную память ($N+4$ целых числа для случая равных вероятностей и $N+1$ вещественных числа дополнительно в случае различных вероятностей присутствия ребер).

Для этого алгоритма доказана следующая теорема. Быстрый алгоритм генерации случайного дерева генерирует любое из N-вершинных деревьев с равной вероятностью $1/N^{N-1}$.

Доказательство довольно очевидно, но утомительно. Сначала доказывается равная вероятность возможных результатов и, затем, доказано что число различных результатов равно N^{N-1}. Поскольку это известное число различных N-вершинных деревьев (включая все перенумерации вершин), достижимость также доказывается. Полное доказательство дано в [?].

Генерация случайных деревьев с ограничением на максимальную степень вершины

Пусть мы имеем ограничение на максимальную степень вершины в дереве: $deg(v_i) \le Deg$. Это ведет к новому определению Ex_i по сравнению с предыдущим случаем:

$$Ex_i = \{\{\, e_{ct} | v_c \in X_i\} \text{если } deg(v_s) \le Deg, \{(e_{ct} | v_c \in X_i) \bigcup (e_{sd} | v_d \in \bar{X}_i)\} \text{иначе.} \qquad (2.40)$$

Добавление новых ребер производится подобно случаю безусловной генерации случайного дерева, но множеств вершин, которые *уже находятся в дереве*, разделен на две части: с максимальной степенью (N_m) и с меньшими степенями ($N_<$). Новое ребро выбирается таким образом, чтобы подключить некоторую свободную вершину с вершиной с из $N_<$. Алгоритм выглядит следующим образом.

Алгоритм генерации случайного дерева с ограничением на максимальную степень вершины (GTD).

Входные данные: `N` – число вершин; `Deg` – возможная максимальная степень вершины.

Выходные данные: `OU[7..N-1]` и `IO[1..N-1]` – векторы номеров начальных и конечных вершин для ребер результирующего дерева (i-е ребро соединяет вершины `OU[i]` и `IO[i]`), `Degrees[1..N]` – вектор для степеней вершин.

```
1.  for i:=1 to N do
2.    I[i]:=i; Degree[i]:=0; \\Начальный вектор номеров вершин
3.  endfor;
4.  m:=1; k:=5; \\m - текущее число вершин дерева;
                \\k - число вершин, запрещенных для дальнейшего
                \\    выбора
5.  l:=random(1,N); Degree[l]:=1; Выбор корня
6.  I[N]:=I[l]; I[l]:=N; \\Перестановка с последним элементом
7.  while m<N do  \\Главный цикл добавления ребер
8.    s:=random(N-m+1,N-k);  \\Для допустимой вершины в дереве
9.    t:=random(1,N-m);      \\выбор новой вершины-соседа
10.   IO[m]:=I[t]; OU[m]:=I[s];  \\Регистрация
11.   INC(Degree[I[t]]); INC(Degree[I[s]]);
12.   if Degree[I[s]]=Deg then \\Проверка на достижение
                               \\максимальной степени
13.     I[s]:=I[N-k]; INC(k);  \\Если ДА, то запретить вершину
                               \\для последующего выбора
14.   endif;
18.   K:=I[t]; I[N-m]:=I[t]; \\Перестановка в рабочем векторе:
      I[t]:=I[N-m];          \\от N-k+1 до N - вершины,
                             \\запрещенные к дальнейшему выбору
16.   INC(m);                \\Завершение рабочего шага
11. endwhile.
```

Замечание: в этом алгоритме не может возникнуть тупик: так как $2 \le Deg \le N-1$, подмножество $X_i^* = \{v_j | v_j \in X_g \& Deg(v_j) < Deg\}$ всегда непусто (по крайней мере последняя вершина, добавлен-

ная к сгенерированному дереву, всегда имеет степень один). А A_{i+1} включает все ребра, которые соединяют эти вершины со свободными и, следовательно, не пусто.

Теперь мы можем просто построить алгоритм для генерации несбалансированного бинарного дерева (B-дерева) с числом вершин $N > 4$. Обратим внимание на то, что степень вершины в B-дереве ограничена 3. Так что мы можем использовать следующую схему генерации (см. Рис. 2):

[h!tb](110,99)(0.20,0.00)bintree.pcx Иллюстрация к алгоритму генерации случайного бинарного дерева

Алгоритм генерации случайного B-дерева.
Входные данные: `N` – число вершин;
Выходные данные: `OU[1..N-1]` и `IO[1..N-1]` – векторы номеров начальных и конечных вершин для ребер результирующего дерева (i-е ребро соединяет вершины `OU[i]` и `IO[i]`), `Root` – номер корневой вершины.

```
1.  N1:=random(1,N+1); N2:=N-N1+1;
3.  Call GTD(N1,3,OU1,IO1,Degree1);
3.  Call GTD(N2,3,OU2,IO2,Degree2);
4.  \\ Используя векторы Degree1 и Degree2 выбираем вершины
    \\ s и t среди вершин степени 1 в обоих деревьях,
    \\ соответственно.
5.  \\ Проводится перенумерация вершин таким образом,
    \\ что вершина s становится N1-ой в первом дереве,
    \\ а вершина t становится первой во втором дереве,
    \\ соответственно.
6.  Root:=N1;     \\ Назначение корня
7.  for i:=1 to N1-1 do
8.    IO[i]:=IO1[i]; OU[i]:=OU1[i];
      \\ Первое дерево составляет левую часть
9.  endfor;
10. for i:=N1 to N1+N2-1 do
11.   IO[i]:=IO2[i]+N1-1; OU[i]:=OU2[i]+N1-1$;
      \\ Второе дерево составляет правую часть
12. endfor.
```

Замечание: если мы принимаем возможность одностороннего B-дерева тогда `N2` может быть 0. В этом случае мы просто генерируем одно дерево с ограничением `Deg=3` и затем выбираем корень из всех вершин со степенью 1 или 2.

Генерация случайного дерева с фиксированными степенями вершин

Теперь мы имеем строгие значения для $deg(v_i)$ в результирующем дереве. Это – более сложный случай, так как может возникнуть тупик при прямом использовании основного алгоритма. Фактически, достаточно чтобы $\sum\limits_j deg(v_j|v_j \in X_i) = 2(i-1)$. Например, если некоторые v_s и v_t обе имеют степени 1, где v_s выбрана в качестве корня на предварительном шаге, а e_{st} – первое выбранное ребро, то мы имеем тупик на самом первом шаге алгоритма. Чтобы избегать тупика, мы используем следующее правило: каждый раз, когда мы имеем $\left(\sum\limits_j deg(v_j|v_j \in X_i) = 2(i-1)-1\right) \& (i < N-2)$, мы добавляем все ребра, которые соединяют вершины из X_i со свободными вершинами с предписанной степенью 5 к Ex_i.

Докажем корректность этого правила, основываясь на доказательстве от противного. Предположим, что $\sum\limits_j deg(v_j|v_j \in X_i) = 2(i-1)-1$ и все свободные узлы имеют предписанную степень 1. Тупик будет иметь место если и только если следующая вершина имеет степень 1. Предположим,

что все свободные узлы (их количество $N-i$) имеют предписанную степень 1. Тогда полная сумма степеней всех вершин равна $2(i-1)-2+N-i=N+i-3$, в то время как для дерева она должна быть $2(N-1)$. Мы имеем противоречие, так что корректность нашего правила доказана.

Генерация случайных деревьев с различными вероятностями существования ребер

Этот случай отличается от предыдущих только в процедуре выбора нового ребра. В случае равной вероятности множество A_i виртуально, и выбор делается между вершинами, теперь же нам необходимо иметь его в некоторой форме, которая позволяет нам делать выбор нового ребра как пронумерованной пары вершин. Отметим, что сумма вероятностей существования ребер не равна 1, поскольку эти события - не альтернативны. Таким образом для выбора *одного* ребра от множества мы используем нормированные вероятности

$$p_i^* = \frac{p_i}{\sum\limits_{j=1}^{n} p_j}. \tag{2.41}$$

Пусть $n = \frac{U(N-1)}{2}$. В дальнейшем мы используем представление неориентированного графа с N вершинами наддиагональною частью его матрицы смежности VV, развернутый в вектор длины n. Мы обозначаем этот вектор как $D(G)$ или просто D. Очевидно имеется взаимно однозначное соответствие между любым элементом VV_{ij} и некоторым элементом D_l. Получим формулу для l через i, j и N.

Каждый элемент VV_{ij} имеет $i-1$ строку выше него и k-ая строка содержит $N-k$ элементов, которые принадлежат наддиагональной части. i-ая строка содержит $j-i-1$ элементов слева от VV_{ij}, которые принадлежат этой наддиагонали части. Таким образом

$$l = [(N-1)+(N-2)+(N-i+0)]+j-i = \frac{2N-i}{2}+j-i. \tag{2.42}$$

Чтобы получить i и j через значения l и N и обратно используем следующие примитивные алгоритмы.

```
1. procedure RowCol(N,l : integer; var Row,Col : integer);
2. var m : integer;
3. begin i:=1;
4.   while (i<N) and (m>0) do
5.     m:=m-N+i; INC(i);
8.   endwhile;
7.   Row:=i-1;$\\
8.   Col:=N+m;$\\
9. end RowCol;
```

```
1. procedure InUpDiag(N,Row,Col : integer}) : inieger};
2. var a,j : integer;
3. begin
4. if Col<Row then a:=Row; Row:=Col; Col:=a; endif;
5. return((2*N-Row)*(Row-1) div 2)+Col-Row;
6. end InUpDiag;
```

Обратим внимание на то, что, если мы не должны экономить память ЭВМ, более просто использовать массив пар (i, j) вместо преобразования, описанного выше. Теперь перейдем к алгоритмам.

Поскольку любая вершина неизбежно принадлежит покрывающему дереву, мы можем выбирать любую в качестве корневой на предварительном шаге с равной вероятностью. В этом случае мы строим начальный вектор допустимых ребер как список ребер, инцидентных корневой вершине.

Просто получить максимальный возможный объем L_m множества допустимых ребер: напомним, что с каждым новым i-м выбранным ребром мы удаляем от этого множества $i-1$ ребро, которые соединяют новую вершину с вершинами из X_i и включаем в него $N-i$ ребро, которые соединяют новую вершину с вершинами из $\bar{X}_i$. Таким образом, после того, как завершается i-й шаг, мы имеем

$$L = (N-1) - 1 + (N-2) - 2 + (N-3) - 3 + \ldots - (i-1) + (N-i) = (N-i)i, \tag{2.43}$$

и $\lceil L_m = N^2/4 \rceil$. Любое дополнительное ограничение может только уменьшить это число. Ниже приводится псевдокод алгоритма генерации в случае дополнительного ограничения на максимальную степень вершины.

Алгоритм генерации случайного дерева с различными вероятностями существования ребер и ограничением на максимальную степень вершины.
Входные данные: $N > 1$ – число вершин, $Prob[1..n]$ – вектор вероятностей существования ребер (этот вектор соответствует $\|\varepsilon_{ij} = P(e_{ij}\text{exists})\|$), Deg - возможная максимальная степень вершины.
Выходные данные: $OU[1..N-1]$ и $IO[1..N-7]$ – векторы номеров начальных и конечных вершин для ребер результирующего дерева (i-е ребро соединяет вершины OU_i и IO_i), $Degrees[1..N]$ - вектор степеней вершин.
Рабочие массивы и переменные: $X[1..N-1]$ – вектор для номеров вершин, которые уже находятся в сгенерированном дереве, $W[1..K]$, $P[1..K]$ – векторы номеров ребер, допустимых для выбора на шаге и их нормированных вероятностей существования, соответственно. M – текущее количество вершин, которые находятся в дереве, и K – число ребер, которые являются допустимыми для выбора на текущем шаге.

```
1.  n:=N*(N-1)/2;  \\ Общее число возможных ребер
2.  \\Инициализация вектора номеров возможных ребер
3.  for i:=1 to n do I[i]:=i; endfor;
4.  \\ Инициализация вектора степеней вершин
5.  for i:=1 to N do Degree[i]:=0; endfor;
6.  l:=random(P,1,n); X[1]:=l; \\Корень выбран
    S:=0.0; j:=0; M:=1; K:=N-1;
    \\ M - текущее число выбранных вершин,
    \\ K - текущее число свободных вершин
7.  for v:=1 to l-1,l+1 to N do
8.    INC(j); W[j]:=I[InUpDiag(j,l)];
    \\ Построение вектора допустимых ребер
9.    S:=S+Prob[InUtDiag(N,v,l)];
    \\ Сумма вероятностей для последующей нормализации
10. endfor;
11. for i:=1 to N-1 do \\ Основной цикл
12.   for v:=1 to K do P[v]:=P[v]/S; endfor;
    \\ Нормализация вероятностей
13.   j:=W[random(P,1,m)]; INC(M); OldS:=S;
    \\ Выбрано новое ребро
14.   Call RowCol(N,j,s,t); INC(Degree[s]); INC(Degree[t]);
    \\ Определены инцидентные вершины
15.   w:=FindPlace(s,X,OK);
    \\ Определим, какая из них новая
16.   if OK then NewNode:=t; else NewNode:=s; s:=t; endif;
    \\ s есть "старая" вершина выбранного ребра
17.   X[M]:=NewNode; \\ Регистрация новой вершины
18.   if Degree[s]=Deg then \\ Проверка на максимальную степень
19.     for t:=1 to N do
    \\ Все свободные ребра инцидентные вершине с
    \\ максимальной степенью должны быть удалены
    \\ из списка допустимых к выбору
20.       w:=FindPlace(t,X,OK);
25.       if not OK then q:=InUpDiag(N,t,s); S:=S-P[q];
          remove(q,W); DEC(K);
22.   endif; endfor; endif;
    \\ Теперь удалим из него ребра, которые соединяют
    \\ новую вершину со "старыми" и включим в него ребра,
    \\ соединяющие новую вершину со свободными
23.   for v:=1 to NewNode-1,NewNode+1 to N do
24.     t:=InUpDiag(X[v],NewNode); w:=FindPlace(t,W,OK);
25.     if OK then S:=S-P[w]; remove(t,W); DEC(K);
26.     else SS:=SS+Prob[t]; INC(K); W[K]:=t;
```

```
27.   endif; endfor;
    \\ Теперь пересчитаем сумму вероятностей для нормализации
28.   S:=SS+S*OldS;
29. endfor.
```

2.6.3 Генерация связных графов

Эта задача – хороший пример алгоритма, в котором правила In_i и Ex_i меняются в процессе генерации.

Каждый связный граф имеет по крайней мере одно покрывающее дерево (единственное, если граф сам есть дерево), так что процесс генерации для связного графа состоит из двух частей: генерация покрывающего дерева и распределения остальных ребер до нужного их количества. Первая задача проанализирована выше. Пусть T – покрывающее дерево, сгенерированное на первом этапе. Если мы не хотим мульти-ребер и ребро e_{st} выбрано на i-м шаге на втором этапе ($i \geq N$), то мы имеем следующие правила для In_i и Ex_i (обратим внимание на то, что $A_{N-5} = E(C(N)) \setminus E(T)$):

$$Ex_i = \{e_{st}\} \tag{2.44}$$
$$In_i = \emptyset \tag{2.45}$$

Если мы должны генерировать связные графы с некоторыми дополнительными ограничениями, то покрывающее дерево должно удовлетворить им, и эти ограничения должны быть приняты во внимание в определении Ex_i.

Генерация случайных связных графов с фиксированными степенями вершин

Если генерацию случайных деревьев с предписанными степенями вершин возможно сделать без тупиков, то в общем случае это невозможно для случайных графов с $M > N$, если мульти-ребра запрещены. Действительно, рассмотрим пример $N = 4$, $M = 5$, $Degrees = (3, 2, 2, 3)$. На первом этапе генерируем случайное дерево, в котором степени вершин *ограничены* данными значениями и, на втором этапе, дополнительные ребра выбираются так, чтобы дать строгое соответствие данным степеням. На Рис. 3 представлен возможный тупик: a) - покрывающее дерево, сгенерированное на первом этапе, b) - единственный правильный вариант для данных степеней, в то время как c) показывает результат основного алгоритма в случае дозволенных мульти-ребер. Ясно, что после выбора первого $(1, 2)$-ребра необходимо выполнить откат.

[h!tb](90,36)(0.00,0.00)ex$_d$*eg.pcx*,

Однако, экспериментально показано, что следующая схема дает меньшее количество откатов в среднем чем универсальная. Пусть мы получили случайное покрывающее дерево на первом этапе алгоритма. Мы имеем $\Delta Deg_i = Degrees[i] - deg(v_i) \geq 0$. Пусть L есть число вершин, для которых неравенство является строгим. Тогда, если

$$\sum_{j=1}^{N} \Delta Deg_i > L(L-1) \tag{2.46}$$

то, чтобы удовлетворить данные степени, необходимы, мульти-ребра и наше покрывающее дерево не подходит, необходим повтор. Иначе мы всегда выбираем вершину с максимумом ΔDeg_i как начальную, в то время как вторая вершина выбирается среди тех, у которых $\Delta Deg_i > 0$.

Очевидно лучше проверять условие 2.46 в процессе генерации покрывающего дерева.

Генерация графов, похожих на реальные сети

Реальные сети имеют много специфических особенностей. В [?, ?, ?, ?, ?, ?] авторы обсуждают различные аспекты структур Internet, Intranet и других сетей. Основываясь на их результатах

и собственном опыте в моделировании различных видов сетей (электросвязи и радио) выпишем наиболее общие свойства коммуникационных сетей.

1) связность;

2) плоскость или "почти плоскость" (если пересечения ребер и имеются, их число невелико);

3) узлы сети расположены в координатной плоскости;

4) неравномерность распределения координат узлов на плоскости: узлы сгруппированы вокруг одного или нескольких центров с понижением плотности с расстоянием;

5) ограничение на минимальное расстояние между вершинами: очевидно, что в в сетях связи слишком короткие расстояния являются столь же невероятными как слишком большие: трудно представить, например, расстояние в несколько метров между двумя базовыми станциями в сотовой сети связи;

6) ограничение на максимальную степень для большинства узлов, или, более точно, вероятность наличия вершин с большим количеством инцидентных ребер мала. В то же самое время в больших сетях существование одной или нескольких вершин с большой степенью весьма вероятно.

Для конкретных классов сетей могут быть добавлены дополнительные свойства и определены количественные значения для названных.

Вероятность существования ребра

Общепринято, что вероятность существования ребра зависит от его длины (расстояния между вершинами) но вид зависимости - это вопрос для обсуждения. В [?, ?], например, вероятность существования ребра между вершинами i и j была выбрана пропорциональной $e^{-d_{ij}}$, где d_{ij} – упомянутое расстояние. Автор рассмотрел более общий случай – пропорциональность

$$p_{ij} \sim e^{-d_{ij}^{\alpha}}. \tag{2.47}$$

На Рис. ?? показаны случайные деревья, полученные алгоритм, рассмотренным в этой главе (100 вершин, максимальная степень вершины – 6) в случае случайных, равномерно распределенных на квадрате координат вершин. На Рис. 4 a) вероятности были рассчитаны с $\alpha = 1$ (т.е. согласно предложению To и Доара), в то время как на Рис. ?? b) и c) вероятности существования ребер рассчитаны пропорциональными $\alpha = 1.5$ и $\alpha = 2$, соответственно. Ясно, что никакая реальная сеть связи не может быть подобна первому дереву, поскольку имеется слишком много пересечений и длинных ребер. Ясно, что второе и третье деревья более подобны реальной сети связи. На Рис. 5 в следующем разделе вероятности также рассчитаны с α равным 1.5 и 2 соответственно. Очевидно, что $\alpha = 2$ более предпочтительный выбор. Однако, в общем случае характер зависимости можно определить только на основе анализа большого количества реальных сетей.

[h!tb](110,37)(0.00,0.00)d$_{11}$5$_2$.*pcx*, e$^{-d_{ij}^{\alpha}}$, $\alpha = 1, 1.5, 2$

Общая схема генерации

В свете рассматриваемых свойств общая схема генерации случайной сети с заданными числом вершин и ребер выглядит следующим образом.

1. Для каждой локальной подсети определяются координаты центральной вершины. При этом добавлено ограничение на минимальное расстояние между этими вершинами. Координаты равномерно определены в пределах данной области (прямоугольник, круг, эллипс и т.д. в зависимостях от конфигурации реальной области. Возможно использовать реальную форму области, но это более трудно для реализации).

2. Согласно заданному (быть может случайному) распределению общего числа вершин между локальными подсетями, определяются координаты этих вершин. При этом рекомендуется

использование двумерного нормального распределения вероятности с независимыми координатами и равными дисперсиями по осям. Величина дисперсии для различных подсетей может быть различна. Повторное испытание требуется при нарушении ограничения на минимальное или максимальное расстояние между вершинами или выходе за границы общей области.

3. Согласно данной убывающей функции от расстояния определяется матрица вероятностей наличия связей между вершинами.

4. С помощью описанного выше алгоритма для генерации случайных деревьев с различными вероятностями существования ребер строится случайное покрывающее дерево. Принимается во внимание ограничение на степень вершины.

5. остающиеся ребра сети выбираются случайным образом согласно данным вероятностям существования и принимая во внимание все ограничения.

Возможен вариант, при котором общее количество ребер заранее распределено при принадлежности к локальным подсетям и соединительным линиям. В этом случае сначала независимо создается каждая подсеть согласно алгоритму, описанному выше, затем случайное дерево, соединяющее эти подсети создается с использованием свободных ребер, соединяющих вершины различных подсетей и, наконец, остающиеся соединительные линии выбираются из этого множества со учетом всех ограничений.

На Рис. 5 и 6 представлены результаты одноуровневой и двухступенчатой генерации графов по предложенной схемой . Координаты вершин – тот же самые, $N = 100,\ M = 150,\ Deg = 6$.

[h!tb](140,65)(0.00,0.00)G1LEV.pcx Случайные графы полученные по одноуровневой схеме генерации. Вероятности определялись по формуле 2.47 с α равным 1.5 и 2 соответственно

[h!tb](140,65)(0.00,0.00)G2LEV.pcx Случайные графы полученные по двухуровневой схеме генерации. а) с центральными вершинами; б) без центральных вершин

Глава 3

Лекции 6-8. Представление систем с дискретными событиями

3.1 Графы событий

Графы событий (ГС) являются удобным средством описания систем с дискретными событиями. Впервые они были предложены в 1983 году Л. Шрубеном [19].

ГС состоит из вершин E_i, соответствующих событиям, и дуг U_{ij}, соответствующих причинно-следственным связям по планированию и отмене событий. В первом случае дуга обозначается сплошной линией, во втором – пунктирной (см. Рис. 1).

Событие определяется изменением переменных состояния системы, происходящим при наступлении этого события. Множество переменных, которое может изменяться при наступлении события E_i обозначим S_i. Переменные состояния могут быть как детерминированными, так и случайными.

Связь событий может быть *условной* и занимать *время срабатывания перехода*. На Рис. 1 видно, что время срабатывания перехода обозначается как вес дуги, а условие задается его номером, стоящим в скобках возле специального значка на дуге. Время t срабатывания перехода по пунктирной дуге от события j к событию k означает,

[h!tb](120,24)(0.00,0.00)ev$_g r1.pcx$

Введем обозначения:

O_i – множество переменных модели, которые могут быть изменены событием i, $O = \bigcup_i O_i$.

E_i – множество переменных модели, которые вовлечены в принятие решений на дугах, исходящих из вершины i, $E = \bigcup_i E_i$.

L_i – множество переменных модели, которые вовлечены в принятие решений внутри правила обработки события i, $L = \bigcup_i L_i$.

E'_i – объединение E_i и L_i, $E' = \bigcup_i E'_i$.

M_i – множество всех переменных модели, используемых событием i, $M = \bigcup_i M_i$. M_i используются в определении условий и вычислении задержек на дугах, исходящих из вершины i.

Рассмотрим простейший пример системы массового обслуживания G/G/1 (Рис. 3.1).

[h!tb](80,25)(0.00,0.00)SMO.pcx Одноканальная СМО

Можно выделит следующие события:

1. Приход нового требования.
2. Взятие требования на обслуживание.
3. Окончание обслуживания.

Граф событий для этого примера представлен на Рис. 3.1
[h!tb](90,70)(0.00,0.00)SMO$_E$G.*pcx*
Используются следующие условия:

(I) – прибор свободен;

(II) – очередь не пуста.

Для проверки этих условий необходимо ввести следующие переменные: `Busy` признак занятости, 0 если свободен и 1 если занят; – `Lq` – текущая длина очереди. Тогда первое условие выглядит как `Busy=0`, а второе как `Lq>0`. Соответственно, имеем E_1 =`Busy`, $E_2 = \emptyset$ и E_3 =`Lq`. Эти переменные меняются при наступлении событий следующим образом. При наступлении 1-го события `Lq` увеличивается на единицу, при наступлении 2-го `Lq` уменьшается на единицу и `Busy` принимает значение 1, и при наступлении 3-го `Busy` принимает значение 0. Таким образом, $O_1 = \{$`Lq`$\}$, $O_2 = \{$`Busy,Lq`$\}$ и $O_3 = \{$`Busy`$\}$.

3.1.1 Применение графов событий

Графы событий позволяют ответить на следующие вопросы:

1. какое подмножество переменных состояния абсолютно необходимо для функционирования модели;
2. присутствуют ли в модели неосуществимые события;
3. какие события должны быть запланированны к моменту запуска модели;
4. можно ли уменьшить количество событий и, соответственно, процедур их обработки.

Рассмотрим по пунктам.

Минимальный набор переменных

Правило 1. Для осуществления правильного перехода от состояния к состоянию достаточно ввести и описать переменные из $E \cup L$. В самом деле, для осуществления безусловных переходов достаточно знать значения задержек на дугах, тогда как верные условные переходы невозможны без знания значений переменных, входящих в соответствующие условия, также как невозможно верное вычисление переменных состояния из O_i без знания значений переменных из L_i. Естественно, при написании имитационной модели могут понадобиться и другие переменные, как для промежуточных вычислений, так и для сбора необходимых для ответов на поставленные вопросы статистик.

Минимальный набор событий, которые необходимо запланировать ДО запуска модели и возможные неосуществимые события

Напомним, что в теории графов *сильносвязной компонентой* ориентированного графа называется подграф, в котором из каждой вершины в каждую существует хотябы один путь.

Удалим из графа событий все пунктирные дуги (дуги отмены событий). Тогда справедиво следующее

Правило 2. Для осуществления правильного функционирования модели необходимо, чтобы хотя бы одно событие в каждой сильносвязной компоненте, не имеющей входных дуг, было запланированно до запуска модели.

Это правило достаточно очевидно. В самом деле, если в компонену нет входных дуг, то возможны лишь внутренние переходы между событиями этой компоненты, управление не может прийти извне. Это означает необходимость начального “толчка"внутри самой компоненты. Если же события в компоненте таковы, что невозможно заранее определить момент наступления ни одного из них, это говорит о *недостижимости* событий этой компоненты и, следовательно, о неверной логике модели.

Приоритеты событий

При выполнении имитационной программы, определяемой некоторым графом событий, может возникнуть ситуация, когда несколько событий должны произойти в один и тот же момент системного времени. Иногда порядок вызова соответствующих процедур безразличен, а иногда разный порядок выполнения даёт различное поведение модели. Для разрешения этой проблемы существует следующее

Правило 3. Если пересечение множеств S_k и E_j непусто, то необходимо установить отношение приоритета для событий k и j.

Редукция графа событий

Будем называть два графа событий эквивалентными, если они при одних и тех же начальных условиях порождают одну и ту же фазовую траекторию в пространстве состояний, т.е. одни и те же изменения переменных состояния во времени. Рассмотрим, когда возможно уменьшить получить эквивалентный граф событий с меньшим числом вершин, уменьшив тем самым число процедур обработки событий.

Правило 4. Эквивалентные графы событий возможны с или без вершины k если эта вершина не имеет исходящих условных дуг и если верно одно из следующих условий:

1. все дуги, входящие в вершину k, имеют нулевое время задержки;
2. O_k не содержит переменных, входящих в условия на дугах;
3. все дуги, исходящие из вершины k, имеют нулевое время задержки.

При выполнении первого условия вершина k может быть объединена с одной из вершин, начальных для рёбер, входящих в неё. Изменения переменных состояния, происходившие в вершине k теперь добавляются к изменениям, проводимым в этой исходящей вершине. Затем вершина k удаляется из графа. Отметим, что в случае применения этого варианта редукции требование на отсутствие исходящих условных дуг не является необходимым.

3.2 DEVS-схемы

DEVS-схемы были предложены Б. Зайглером [B. Zeigler] для решения следующих задач:

1. построение теоретической базы для доказательства корректности и эквивалентности имитационных моделей;
2. предоставления инструмента для описания иерархически организованных моделей, удобного для организации хранения компонент и распределънного исполнения моделирующей программы.

Название представляет собою аббревиатуру от “Discreete EVent System (Система с дискретными событиями)”. Концептуально этот подход восходит к агрегированным схемам Н.П. Бусленко, но более строго формализован.

Основой рассматриваемого представления является понятие DEVS-компоненты, которая представляет из себя шестёрку

$$COMP =< S, X, Y, \delta, \lambda, ta >, \tag{3.1}$$

где

S – множество последовательных состояний;

X – множество внешних входных событий;

Y – множество выходных событий;

λ – выходная функция от состояния, производящая выходные события ($\lambda : S \to Y$);

ta – функция продвижения времени, отображающая множество состояний на множество неотрицательных вещественных чисел ($S \to R^{+}_{0,\infty}$);

δ – функция смены состояния состоящая из двух частей, δ_{int} и δ_{ext}. δ_{int} применяется при отсутствии внешних воздействий в течение времени $ta(s)$ по истечению этого периода, при этом время нахождения компоненты в текущем состоянии e сбрасывается в 0 ($\delta_{int} : S \to S$). δ_{ext} применяется, когда внешнее событие наступает прежде истечения периода времени $ta(s)$, e также сбрасывается в 0 ($\delta_{ext} : X \times Q \to S$). Здесь Q есть обобщенное состояние, определяемое как

$$Q = \{(s, e) | s \in S \text{ and } 0 \leq e \leq ta(s)\}. \tag{3.2}$$

Тем самым COMP представляет собою компоненту системы, которая подвергается воздействиям входных событий X, производимых окружением, и которая, в свою очередь, производит выходные события Y, которые воздействуют на окружение.

Состояние s компоненты модели вычисляется соответствующей функцией перехода: внутренней δ_{int} или внешней δ_{ext}. В дискретно-событийной модели планируется наступление “внутреннего"события через $ta(s)$ единиц времени. Если в момент времени e этого периода ($e \leq ta(s)$) происходит входное событие, то посредством δ_{ext} определяется новое состояние s' и определяется новое время $ta(s')$ до очередного “внутреннего"события. Иначе, когда выполнится $e = ta(s)$, определяется выходное событие компоненты как $Y = \lambda(s)$ и посредством δ_{int} определяется новое состояние компоненты s' и определяется новое время $ta(s')$ до очередного “внутреннего"события. Если выходное событие должно быть произведено как результат происшествия входного, то это означает, что после пересчета состояния функцией δ_{ext} $ta(s') = 0$.

Для связи компонент DEVS в модели используются так называемые *связующие* компоненты (в английском оригинале – generic components). Таких компонент 6 и они представлены на Рис. **??**.

[h!tb](120,120)(0.00,0.00)generic.pcx Связующие компоненты DEVS

Эти компоненты выполняют следующие функции:

$SYNC(X_1, X_2, \ldots, X_n, Y_1, Y_2, \ldots, Y_n)$ – синхронизация событий: события, произошедшие асинхронно на входных полюсах, одновременно повторяются на выходных полюсах в момент происшествия последнего из входных. Если на входном полюсе произошло новое событие до момента выдачи выходных, оно будет ожидать поступления следующих (не очередных) событий на других полюсах и повторится на выходе в следующий раз и т.д.

$DELA(X, Y, d)$ – повторение входного события на выходе по истечению заданного периода времени d.

$ABST(X_1, X_2, \ldots, X_n, Y)$ – формирует единственное выходное событие Y как кросс-произведение входных событий $X_1, X_2, \ldots, X_n$.

$PROJ(X, \bar{X}_1, \bar{X}_2, \ldots, \bar{X}_m, Y_1, Y_2, \ldots, Y_n, E_1, E_2, \ldots, E_n)$ – декомпозирует единый вход X на компоненты $Y_1, Y_2, \ldots, Y_n$. Вход X состоит из $\bar{X}_1, \bar{X}_2, \ldots, \bar{X}_m$ и $Y_i = \underset{j \in E_i}{\times} \bar{X}_j$, где $E_i \subset \{1, 2, \ldots, m\}$.

$SELE(X_1, X_2, \ldots, X_n, Y)$ – выбирает одно из входных событий X_i в качестве выходного события, т.е. $Y = X_i,\ i \in \{1, 2, \ldots, n\}$.

$REPL(X, Y_1, Y_2, \ldots, Y_n, E)$ – повторяет входное событие X на каждом выходе, специфицированном параметром E, где $E \subset \{1, 2, \ldots, n\}$.

Несколько компонент DEVS, соединенных посредством связующих компонент, могут образовать *мультикомпонент* DEVS.

Определение Мультикомпонент DEVS есть пятерка

$$MULC =< \{SET.COMP\}, \{SET.GEN\}, \{IN\}, \{OUT\}, SELECT >, \tag{3.3}$$

где

SET.COMP – множество DEVS COMP;

SET.GEN – множество связующих компонент DEVS;

для каждой компоненты i из SET.COMP

IN – множество компонент COMP, которые влияют на i (*влияющие*);

OUT – множество компонент COMP, на которые влияет i (*влияемые*);

IN,OUT $\in$ **SET.COMP** ;

SELECT – функция развязывания узлов.

В мультикомпоненте DEVS в качестве внутренней функции смены состояния выбирается соответствующая функция той ее компоненты, запланированное время выполнения которой минимально. Если таких компонент несколько, то выбор делается с помощью функции SELECT, реализующей некоторую схему приоритетов. *Связующие компоненты всегда обладают максимальным приоритетом.*

[h!tb](125,80)(0.00,0.00)SimpleCS Пример DEVS-модели простой вычислительной системы

Рассмотрим пример мультикомпонента. На Рис. 3.2 представлена модель простейшей вычислительной системы. Схема её работы ясна из пояснений на рисунке. Определим эту модель как мультикомпонент DEVS. Согласно приведённому выше описанию имеем:

- $SET.COMP = \{INPUT\_Q, CPU, MEMORY\}$;
- $SET.GEN = \{PROJ_1, PROJ_2, ABST_1, ABST_2, ABST_3, SYNC, DELA\}$;
- $IN(INPUT\_Q) = null$;
- $IN(CPU) = \{INPUT\_Q, MEMORY\}$;
- $IN(MEMORY) = \{INPUT\_Q, CPU\}$;
- $OUT(INPUT_Q) = \{CPU, MEMORY\}$;
- $OUT(CPU) = \{MEMORY\}$;
- $OUT(MEMORY) = \{CPU\}$.

Функция $SELECT$ задаёт следующую схему приоритетов:

$$\{generic\ components\} > INPUT_Q > MEMORY > CPU. \tag{3.4}$$

Можно доказать, что мульти-компонента $MULC$ сама является компонентой DEVS COMP, то есть для неё возможно определить шестёрку $< S, X, Y, \delta, \lambda, ta >$.

При исполнении модели, представленной DEVS-схемой, необходимо уметь декомпозировать входящие в неё мульти-компоненты, поскольку исполняемые функции смены состояния определены непосредственно только для простых компонент. Для сворачивания-разворачивания DEVS-схемы существует алгоритм, представленный на Рис. 3.2.

[h!tb](125,70)(0.00,0.00)Topdown Алгоритм разбора-исполнения DEVS-модели

Псевдокод представленных на этом рисунке процедур имеет следующий вид

```
Algorithm: DECOMPOSE
Input    : DEVS COMP
Output   : DEVS hierarchy
Method   :
Procedure DECOMPOSE(DEVS COMP) : returns DEVS MULC
1.  initialize queue to empty
2.  loop
3.    determine the components of DEVS COMP
4.    for all output ports, i, of components of DEVS COMP
5.      call DISTRIBUTE(output line_i)
6.    end for
7.    call COUPLE(set of DEVS components)
8.    call CLOSE(X,Y)
9.    store the indexes of components in queue
10.   while component is not to be decomposed and queue not empty
11.     remove component at front of queue, call it COMP
12.   end while
13.   if COMP is not to be decomposed and queue is empty then exit
14. forever
15. end DECOMPOSE

Algorithm: DISTRIBUTE
Input    : output port
Output   : output port(s) with generic components
Method   :
Procedure DISTRIBUTE(DEVS Y) : returns DEVS Y
1.  if there is delay in the output port, then insert a DELA
2.  determine how the output port will be distributed:
3.    case 1: if the output port will influence only one COMP,
              then do nothing
4.    case 2: if the output port will influence more than one
              COMP, then connect the output port to a REPL and
              label each output port from the REPL as as
              Y_1,Y_2,...,Y_n, where Y_i=Y.
              for each Y_i, if delay is part of the model, then
              insert a DELA.
5.    case 3: if the output port will be decomposed into subsets
              more than one COMP, then connect the output port to
              a REPL and label each output port from the REPL as
              Y_1,Y_2,...,Y_n, where Y_i=Y.
              for each Y_i, if delay is part of the model, then
              insert a DELA.
6.    end case
7.  end DISTRIBUTE

Algorithm: COUPLE
Input    : Set of DEVS Components
Output   : DEVS MULC
Method   :
Procedure COUPLE(Set of DEVS Components) : returns DEVSMULC
1.  for each component, i, in Set
2.    obtain the set of influencers, IN(COMP_i)
```

```
3.    for each COMP_j, where j is in IN(COMP_i)
4.      determine the type of relationship with COMP_i
5.      case 1: if Y_j has to wait for other output events to
              COMP_i, then connect the output port_j to a SYNC
6.      case 2: if Y_j is to be one of other output events to
              COMP_i, then connect the output port_j to an ABST
7.      case 3: if Y_j is to be selected from among several
              output events to COMP_i, then connect the output
              port_j to a SELE
8.      end case
9.      if there are more than one input events to COMP_i, then
        insert an ABST
10.   end for
11. enf for
12. end COUPLE

Algorithm: CLOSE
Input    : DEVS MULC,X,Y
Output   : DEVS COMP
Method   :
Procedure CLOSE(DEVS MULC,X,Y) : returns DEVS COMP
1.  call DISTRIBUTE(X) and connect
2.  bundle all output port(s) whose destination is
    an external environment
3.  insert an ABST and call the output port, Y
5. end CLOSE
```

Глава 4

Лекции 9-10. Вопросы программной реализации систем дискретно-событийного моделирования

В этих лекциях рассматриваются способы синхронизации событий в программах имитации дискретных систем. Особое внимание обращается на приемы ускорения обработки календаря событий за счет использования групп событий и синхронизации в распределенных сигнало-ориентированных системах моделирования. Выделяется рассмотрение такого важного случая групп событий, как одновременные.

Основные собственные результаты исследований по этим вопросам опубликованы в [?, ?, ?, ?, ?, ?, ?, ?].

4.1 Основные понятия и обозначения

Для определения основных понятий с небольшими изменениями воспользуемся [10].

Под *системой* понимается:

$$S =< T, X, \Omega, Q, Y, \delta, \lambda >, \tag{4.1}$$

где

T – временная база, множество моментов времени, служащее для упорядочения событий. Обычным выбором служат I^+ или R^+. Соответственно говорят о системах с дискретным или непрерывным временем. О системах с дискретным временем говорят и тогда, когда T есть дискретное подмножество R^+.

X – множество входных значений, то есть множество, представляющее возможное влияние окружающей среды на систему. Вообще говоря, произвольного вида. Будем считать, что обязательной компонентой входа является $t_c \in T$ – текущее модельное время.

Ω – множество входных сегментов, описывает хронологическую картину входных воздействий на систему за некоторый отрезок времени. Иными словами, входной сегмент есть отображение $\omega :< t_i, t_f > \rightarrow X$, где t_i и t_f есть соответственно начало и конец отрезка временной базы. Вид Ω определяется выбором временной базы.

Q – множество внутренних состояний, представляет собой память системы, то есть ту часть истории ее поведения, которая может повлиять на поведение в дальнейшем. Вообще говоря, произвольного вида.

Y – множество выходных значений, то есть множество, представляющее возможное влияние системы на окружающую среду. Вообще говоря, произвольного вида.

δ – функция смены состояния, отображение $\delta : Q \times \Omega \to Q$, то есть она определяет для системы в состоянии q в момент времени t_i при приложении входного сегмента $\omega :< t_i, t_f >\to X$, что она будет находиться в состоянии $\delta(q, \omega)$ в момент времени t_f. Требование того, что множество состояний Q должно содержать всю необходимую информацию о прошлом, усиливается требованием к функции δ удовлетворять полугрупповому свойству, то есть для каждого $q \in Q, \omega \in \Omega$ и t из домена $\omega, < t_i, t_f > * \delta(q, \omega) = \delta[\delta(q, \omega_{t>})q, \omega_{t<}]$,

где $\omega_{t>} = \omega| < t_i, t >$ (часть ω между t_i и t), а $\omega_{t<} = \omega| < t, t_f >$ (часть ω между t и t_f). Дискретное изменение состояния системы, происходящее мгновенно в момент времени t_i, назовем **событием** или **дискретным событием**.

λ – выходная функция, отображение $\lambda : Q \to Y$. Иными словами, $\lambda(q)$ представляет собой то, что окружение (наблюдатель) может получить (измерить) от системы, когда она находится в состоянии q.

В случае рассмотрения систем с *дискретными событиями* будем различать O-**события**, определяемые внешними воздействиями на систему, и I-**события**, определяемые внутренними изменениями состояния системы, когда функция δ использует только временную составляющую Q.

4.2 Программные модели систем

При имитационном моделировании формальной системе, описанной в предыдущем пункте, ставится в соответствие программная модель, реализующая ее поведение. При построении программной модели естественно происходит некоторое огрубление модели, связанное с ограничениями выбранных аппаратно-программных средств.

Рассмотрим соответствие элементов формального описания системы элементам программной модели.

- временной базе соответствует целочисленная или вещественная переменная неубывающая за время исполнения программы (далее называемая *time*).
- множества входных значений, внутренних состояний и выходных значений в соответствиями с возможностями выбранной системы программирования могут включать целые и вещественные переменные, массивы, записи, указатели и переменные всех других доступных типов.
- множество входных сегментов есть последовательность входных значений, упорядоченная в соответствии со значением переменной *time*.
- функция смены состояния, в соответствии с выбранным способом представления динамики модели, реализуется последовательным вызовом процедур F_i и/или передачей управления между процессами (сопрограммами) P_i обработки событий. Для организации передачи управления между процедурами и/или процессами используется специальная *программа обработки календаря событий (управляющего списка)* от эффективности реализации которого во многом зависит эффективность исполнения программной модели.
- выходной функции соответствует некоторое подмножество множества внутренних состояний или функция от него.

Рассмотрим подробнее обработку **событий**. В общем случае в имитационном моделировании дискретных систем под событием понимают пару $< time, \delta >$, где $time$ – момент *системного времени*, а δ – функция *смены состояния* модели, вызов некой процедуры F_i или передача управления некоторому процессу P_i в зависимости от выбранной системы моделирования. Однако часто, в частности благодаря событийно-ориентированным языкам моделирования, понятие события отождествляется с конкретной функцией смены состояния. В этом случае говорят о событии E произошедшем в момент времени $time$. В данной лекции мы будем придерживаться именно этого определения. В случае, когда требуется понимать событие согласно первому определению, будем говорить о *событии в узком смысле*.

Итак, *Функция смены состояния может быть реализована с помощью процедуры или активной фазы процесса (сопрограммы)*.

Следующим важным понятием является понятие **класса** событий. Оно сходно с понятием класса объектов: под классом событий будем понимать множество событий в узком смысле, требующие для своей обработки запуска одной и той же функции смены состояния модели (возможно, с различными параметрами).

Календарем событий или *управляющим списком* L_c будем называть упорядоченный по модельному (системному) времени список *уведомлений о событиях*, где уведомление о событии есть тройка $E_{time} =< time, F_i[P_i], next >$, где $next$ есть ссылка на следующее в списке уведомление о событии (Рис. 4.2). Далее, для удобства, если не требуется делать специального различия, вместо $F_i[P_i]$ будем использовать обозначение FP_i для обработчика события. На практике, что и нашло отражение на рисунке, уведомление о событии содержит ряд дополнительных полей (например, текстовое имя события), которые могут использоваться для отладки (режим трассировки событий) или более эффективного поиска по календарю.

[h](100,39)(0.00,0.00)Calendar Стандартная структура календаря событий (управляющего списка)

Управляющий алгоритм системы моделирования заключается в последовательном вызове обработчиков событий FP_i с одновременной сменой значения системной переменной **время** на $time_i$.

4.3 Способы планирования событий

В случае *императивного планирования* событий время наступления события E_i определяется с помощью операторов вида

“планировать событие E_i на время τ_i”.

Тем самым все O-события планируются императивно. Выполнение каждого такого оператора планирования требует проведения поиска по управляющему списку для определения места включения в него нового уведомления о событии. Основным приемом сокращения времени поиска является применение нелинейных списков, например, бинарных сбалансированных деревьев [12].

В случае *интеррогативного планирования* событий время наступления события E_i определяется с помощью операторов вида

“E_i наступает при условии condition”.

Подобные операторы часто бывают удобны для планирования I-событий. Их использование в программных моделях требует специального аппарата проверки условий наступления событий. Качество реализации этого аппарата весьма существенно влияет на эффективность программ моделерования [13, 14].

Нужно отметить, что особую сложность реализация аппарата планирования имеет при процессно-ориентированном подходе к описанию моделей, когда в правиле действий процесса встречаются операторы остановки до наступления момента выполнения некоторого условия.

4.3.1 Реализация оператора WAIT_UNTIL в процессно-ориентированных системах моделирования

Поскольку, в принципе, выполнение условия продолжения (срабатывание условия) возможно после каждого события, необходимо проверять список процессов, остановленных до выполнения условий, после отработки каждого события. Проще всего сделать это введением специального процесса проверки условий [13], который после однократного внешнего планирования будет автоматически перепланировать себя после каждого события, связанного с другими процессами.

Если мы имеем дело с непрерывно-дискретным моделированием и в условиях присутствует сравнение непрерывно изменяющихся величин, тогда приходится проверять условия продолжения в ходе изменения этих величин. Если делать это с максимальной возможной точностью, то есть с шагом интегрирования (считаем, что непрерывная составляющая модели описывается системой дифференциальных уравнений), эффективность исполнения программы модели резко снизится. Если же проверять эти условия с некоторым бóльшим шагом, называемым шагом проверки условий [12], возникает опасность пропуска нужного события, что накладывает дополнительную ответственность на разработчика модели. Можно отметить также, что проверка условий продолжения в ходе процесса интегрирования требует использования продвижения модельного времени с шагом интегрирования, то есть использования специального процесса интегрирования, перепланирующего себя с этим шагом, что крайне неэффективно. Это показал, в частности, отрицательный пример класса `DisCont` моделирования непрерывно-дискретных систем на Симула-67, предложенного в [12], который при удобстве описания моделей давал очень медленные в исполнении программы.

Снизить затраты на проверку условий (но не надежность обнаружения их выполнения) можно программируемым запуском процесса проверки после событий заданных классов [14] и/или на заданных временных отрезках (в результате прогноза), то есть использованием операторов вида

“`WAIT_UNTIL condition ON time_interval`”.

Продолжение работы процесса в этом случае произойдет либо по выполнению условия после начала указанного временного интервала, либо по истечении этого интервала.

В существующих языках моделирования, как правило, время действия оператора ожидания либо не ограничено, либо ограничено только сверху [18, 8, ?].

Хорошее повышение эффективности исполнения программ в случае отсутствия в условиях задержки непрерывных переменных дает [14] группирование задержанных процессов по принципу возможности продолжения после тех или иных событий в модели и, соответственно, запуск процедуры проверки условий в программах обработки этих событий. Следовательно, требуется оператор остановки процесса вида

“`WAIT_UNTIL condition IN set`”

и процедура вида

“`TestConditionFor(set)`”

4.3.2 Отложенные события в событийно-ориентированных системах моделирования

При схожести проблем, событийно-ориентированные системы моделирования требуют несколько иной реализации средств интеррогативного планирования событий. Соответствующий оператор планирования в таких системах принимает вид

“`SCHEDULE event WHEN condition`”,

что означает вызов программы обработки события *event* в момент выполнения условия *condition*.

Более сложной формой оператора является

“SCHEDULE event EACH TIME WHEN condition”,

что означает вызов программы обработки события *event* каждый раз, когда выполнится условие *condition* (условный генератор событий).

Как и в процессно-ориентированных системах моделирования резко возрастают затраты на проверку выполнения условий при наличии в модели непрерывно меняющихся величин, входящих в условия. Вместо процесса проверки условий, о котором говорилось в предыдущем пункте, приходится вводить событие исполнения процедуры проверки, частота вызова которого снижает скорость исполнения модели и повышает точность и надежность определения момента отложенных событий. Методы повышения скорости исполнения программ аналогичны рассмотренным выше.

4.4 Использование групп событий для повышения эффективности моделирования

Вполне очевидно что существенную роль в эффективности систем моделирования играет реализация календаря событий. Если внешне этот календарь представляет собою линейный, упорядоченный по времени запланированного наступления событий список, то внутренняя его организация может быть много сложнее. так, очевидно, реализация его в виде сбалансированного дерева будет много более эффективна при большом объёме календаря. Ниже мы рассмотрим более простые в реализации способы использования групп событий, позволяющие организовывать календарь в виде двухуровневого списка (списка списков). В этом случае в основном списке присутствует единственный, ближайший по времени наступления, представитель каждого списка второго уровня. При наступлении этого события ищется место в основном списке для следующего представителя соответствующего списка второго уровня. Предлагаемые группы событий позволяют организовывать описанную схему наиболее естественным образом и с минимальной трудоёмкостью программирования.

Для повышения эффективности реализации систем имитационного моделирования имеет смысл выделять следующие виды групп событий.

4.4.1 Строго последовательные цепочки

Данная группа представляет собою строго упорядоченное множество событий $\{E_1, E_2, \ldots, E_n\}$, такое, что в рамках строго последовательной цепочки (СПЦ) событие E_i может наступить только после события E_{i-1}, причем через, вообще говоря, случайное время $t_i = time_i - time_{i-1}$. Событие E_i в общем случае может принадлежать произвольному количеству цепочек, но не одновременно.

[h!tb](102,98)(0.00,0.00)SPC.bmp Управляющий список с учетом СПЦ

Выделение СПЦ позволяет первоначально включать в управляющий список только событие E_1 а затем, после обработки очередного E_i планировать E_{i+1}, где $i = 1, \ldots, n-1$ (рис. 2.2).

4.4.2 Одновременные события

Ряд событий может произойти в один и тот же момент модельного времени, образуя группу одновременных событий (ОС). Различают упорядоченную и неупорядоченную группы ОС. Логически упорядоченная группа ОС не отличается от строго последовательной цепочки, но имеет иную программную реализацию.

Вообще говоря, случай одновременных событий соответствует редуцируемой группе вершин в графе событий [19], то есть для обработки этих событий возможно написать одну специальную процедуру. Однако это может быть неудобным при наличии готовых процедур обработки для некоторых из событий, входящих в группу.

[h!tb](102,98)(0.00,0.00)odnoim.bmp

Управляющий список с группой одновременных упорядоченных событий

При наличии одновременных событий целесообразно предусмотреть двухуровневую организацию управляющего списка (рис. 2.3). В основной управляющий список включается первое из одновременных событий. При осуществлении этого события, по окончании его обработки последовательно обрабатываются остальные ОС данной группы.

4.4.3 Одноименные события

Под одноименными событиями мы понимаем события, требующие для своей обработки одной и той же программы (возможно, с разными параметрами).

Планирование и исполнение одноименных событий также целесообразно организовывать по двухуровневому принципу: для одноименных событий при описании заводится список, в который будут заноситься уведомления об этих событиях, аналогичные уведомлениям основного календаря. Список упорядочен по времени запланированных событий. По мере продвижения модельного времени в основной список переносятся уведомления с наименьшим временем запланированного события. При планировании события производится проверка: не меньше ли время его вызова, чем запланированное время события группы, присутствующего в основном календаре. Если да, то присутствующее в календаре уведомление вытесняется на первое место во вспомогательный список, а поиск места для нового уведомления ведется в основном календаре. Если нет, то поиск места ведется во вспомогательном списке. [h!tb](102,98)(0.00,0.00)odnoim.bmp Управляющий список с учетом одноимённых событий

4.4.4 Отложенные события, происходящие при наступлении одного и того же условия

Отложенные события, происходящие при наступлении одного и того же условия, являются аналогом одновременных событий при интеррогативном планировании. В момент выполнения условия последовательно вызываются программы обработки событий группы.

4.5 Управление событиями в транзактно-ориентированных системах моделирования

В транзактно-ориентированных системах моделирования (наиболее известный представитель – GPSS [?, ?]) можно использовать специфику некоторых классов событий для повышения эффективности моделирования.

4.5.1 События, связанные с использованием устройств

Блоки работы с устройствами или ресурсами (`SIZE - RELEASE, ENTER - LEAVE, PREEMPT - RETURN`) вырабатывают специфические события: требование на занятие некоторого ресурса и его освобождение. В случае невозможности выделения требуемого ресурса процесс продвижения транзакта останавливается на неопределенное время, до появления возможности его выделения.

В структурах, соответствующих устройствам и ресурсам, организуются цепи задержанных транзактов (обычно в порядке поступления заявки). В отличие от общего случая событий, отложенных до выполнения условий, проверять условие возможности занятия устройства нет необходимости, поскольку имеется явная функция его освобождения, которая, в частности, проверяет наличие очереди к этому устройству и, если она имеется, разрешает продвижение соответствующего транзакта. Проверка возможности занятия части разделяемого ресурса также запускается явно при обработке события освобождения ранее занятой части. Календарь событий в этом процессе не участвует.

4.5.2 События, связанные с блоком задержки транзактов ADVANCE

При поступлении транзакта на блок задержки `ADVANCE` определяется время задержки и, соответственно, время события выхода из блока. В общем случае случайный характер задержки может привести к изменению порядка выхода транзактов из блока по сравнению с порядком входа в него.

Возможно организовать отдельный список задержанных транзактов для всех блоков ADVANCE, сохраняя в нем транзакты с номерами блоков, на которые они должны поступать по окончании задержки, либо организовать отдельные списки для каждого блока, но уже без этих номеров (блок`ADVANCE`имеет один выход). Поиск по списку необходим и в том и в другом случае, при первом варианте в основном календаре событий может находится одно уведомление (как и для одноименных событий, частный случай которых и рассматривается), во втором – по одному на каждый блок ADVANCE. Преимущество того или иного подхода определяется количеством блоков`ADVANCE` и интенсивностями потоков транзактов через них.

В [?] был предложен также вариант введения специального блока задержки на постоянную величину, что часто имеет место в моделях дискретных технических и информационных систем. Введение такого блока при втором варианте позволяет исключить операцию поиска места в списке задержанных транзактов, поскольку порядок их следования не нарушается и очередной транзакт ставится в список последним.

Группы событий в моделях ненадежных систем

В качестве примера класса моделей, в которых естественным образом возникают группы одновременных событий можно рассматривать ненадежные технические системы (системы с вероятными отказами компонентов), например, цифровые сети интегрального обслуживания, системы управления реального времени и другие аппаратно-программные системы (АПС). Действительно, в таких системах отказ одного компонента может повлечь остановку многих процессов (выход из строя узла коммутации приводит к прерыванию большого количества сеансов связи, отказ аппаратного элемента может повлечь за собою прекращение выполнения всех программ, его использующих), а затем, после восстановления компонента - возобновление их работы. При использовании классического календаря событий и соответствующих процедур планирования и отмены событий придется в цикле осуществлять многократный вход в календарь событий для включения или удаления уведомлений. Иерархическая организация календаря позволяет в этом случае существенно сократить время поиска.

В случае применения операторов ожидания условия также могут возникнуть ситуации, когда одного и того же условия ожидает большое количество процессов (много программ ожидают восстановления аппаратуры). В этих условиях нецелесообразно осуществлять проверку условия для каждого из ожидающих процессов, гораздо эффективнее проверять его для всей группы сразу.

В качестве расширения пакета моделирования СИДМ-2 [?] разработан специальный модуль `GrSchedule`, реализующий процедуры группового императивного управления событиями:

`GrActivate` – планирования событий для группы объектов на один и тот же момент модельного времени (в порядке перечисления),

`GrCancel` – отмены запланированных событий для группы объектов,

`GrDelay` – задержки событий на заданное время для группы объектов,

`GrRestart` – аналогична процедуре GrActivate, но работа процедур объектов начинается от начала.

4.6 Синхронизация событий в распределенных системах ДИМ

Задача распределенной имитации возникает при необходимости проведения масштабных экспериментов при отсутствии высокопроизводительной ЭВМ. В этом случае распараллеливание процесса имитации осуществляется на локальной сети. Другой возможной причиной распределения имитации может служить приближение исполняемых процессов к используемым ими большим массивам

данных (проще переслать на исполнение относительно небольшую программу и затем получить ее результат, чем пересылать большой объем данных). Для моделирования цифровых сетей существует еще одна причина для распределения: создание полунатурных моделей, когда часть системы реально функционирует, а другая представлена моделями компонент.

Предметом нашего исследования являются способы представления моделей для распределенной имитации и методы управления ими.

4.6.1 Основные проблемы осуществления распределенной имитации

Часть проблем, связанных с осуществлением распределенной имитации, является общей для распределенных вычислений вообще. Прежде всего это проблемы запуска, останова и обработки ошибочных ситуаций.

Запуск. Запуск модели, вообще говоря, имеет смысл только в том случае, когда все компоненты расположены на соответствующих узлах сети и готовы к исполнению. Соответственно, должна существовать некоторая программа подготовки модели к запуску, которая осуществляет распределение объектов по компьютерам сети (при этом часть компонент привязана к конкретным компьютерам), получает подтверждение о готовности от каждого компьютера и рассылает сигнал запуска модели в работу. С другой стороны, возможен вариант, когда некоторая компонента модели необходима не сразу, а спустя некоторое время работы модели (например, программа документирования результатов). В этом случае возможны распределение и запуск некоторого начального набора компонент с последующим динамическим подключением остальных. Этот вариант, очевидно, предпочтительней с точки зрения используемых ресурсов, но сложнее в реализации.

Останов. Возможны варианты останова моделирования по завершению: всех компонент, любой компоненты, выделенной компоненты. В первом случае завершение работы объекта наступает вследствие достижения его состояния некоторой критической области и последней должна завершиться работа компоненты вывода результатов. Во втором случае программа работы объекта должна завершаться вызовом специальной программы рассылки сообщения об окончании работы всем объектам, либо посылки такого сообщения некоторому "главному"объекту, который, в свою очередь, разошлет сообщение об окончании остальным. В третьем случае рассылка таких сообщений предусматривается в программе этого "особого"объекта.

Обработка ошибочных ситуаций. В случае распределенного исполнения ошибка может возникнуть либо в отдельной компоненте, либо в системе связи компонент. Нарушения в системе связи фатальны и должны привести к останову модели. При сбое в компоненте возможны варианты: останова модели, перезапуска компоненты, перезапуска компоненты на другом компьютере, уничтожения компоненты с перераспределением составляющих ее объектов между остальными компьютерами. Варианты с перезапуском и перераспределением достаточно сложны в реализации но повышают живучесть моделей. Проблему представляет также само обнаружение сбоя: поскольку сбой может быть таким, что компонента не сможет сообщить о нем (например, вследствие "зависания"компьютера), то его обнаруживает другая компонента по невозможности "достучаться"до сбойной. Необходима система критериев, позволяющих определить состояние нереагирующих на посылаемые сообщения компонент.

4.6.2 Синхронизация событий

Наибольший интерес с точки зрения эффективности и безошибочности моделирования, однако, представляет организация механизма синхронизации дискретных событий, происходящих в распределенных по сети компонентах. Этому вопросу в последние годы посвящено множество публикаций, например [?, ?].

Наиболее просто реализуются модели с синхронизацией с помощью единого календаря событий и соответствующей управляющей программы. Однако наибольшей эффективности и устойчивости можно достигнуть при асинхронном выполнении программ компонент, которые имеют свои локальные календари. В этом случае необходимо уметь производить "откат"при поступлении сигнала, да-

тированного прошедшим временем, равно как организовывать очередь на обработку "сигналов из будущего".

В [?] автором предложена модификация известного алгоритма с откатом Д.Берта. В модификации используется следующая теорема этого же автора:

Следующие утверждения равносильны:

- Существует такое значение t, что все $t_j > t$, где t_j есть модельное время j-й компоненты;

- Не существует сообщений или антисообщений с меткой времени меньшей чем t.

Здесь под антисообщением $S^* = (time, Par, i, j)^*$ понимается сообщение об отмене сообщения $S = (time, Par, i, j)$. Каждое сообщение имеет вектор полностью его идентифицирующих параметров Par и метку времени $time$, равную моменту времени его посылки подмоделью (компонентой) C_i на подмодель C_j.

Модель представляется в виде орграфа подмоделей C_i. Дуга (i, j) соответствует возможности посылки сообщений от i-й к j-й подмодели. Каждой дуге соответствует буфер B_{ij} в котором соответствующие сообщения хранятся в порядке роста меток времени. Вершина C_0 считается особой (управляющей). В каждой подмодели хранится следующая информация:

1. Некоторое число t_i, интерпретируемое как наибольшее значение времени t такое, что состояние подмодели $Xi(t)$ было вычислено и не было отменено с помощью отката.

2. Список $LX(i)$, содержащий значения состояний подмодели для всех t на интервале от $t\_$ до t_i. Здесь $t\_$ – некоторое значение времени, определяемое алгоритмом.

3. Список $LM(i)$, содержащий все сообщения, которые были приняты, обработаны и на которые не были получены антисообщения.

4. Список $L^*M(i)$, содержащий все антисообщения, которые были приняты, обработаны и на которые не были получены сообщения.

5. Список $LS(i)$, содержащий все сообщения, которые были посланы и для которых не были посланы антисообщения.

Вершина C_0, кроме того, имеет вектор T длины $N - 1$, где N - число вершин.

Первоначально все списки пусты и t_i имеет произвольное отрицательное значение. Вектор T вершины C_0 заполнен отрицательными значениями. Далее алгоритм работает следующим образом:

Шаг 1. Если сообщение $(time, Par, j, i)$ есть в одном из буферов B_{ji}, удаляем его оттуда и делаем следующее:

1.1. Если $i = 0$ и $T_j < 0$ или $T_j > time$, то $T_i = time$.

1.2. Если антисообщение $(time, Par, i, j)^*$ уже принято в $L^*M(i)$, уничтожить и сообщение и антисообщение.

1.3. Если антисообщение не найдено, поместить сообщение в $LM(i)$ и если $time < t_i$, то произвести откат:

1.3.1. (Откат) Принять $T_i = time$. Исключить из $LX(i)$ $Xi(t)$ для всех $t > time$.

1.3.2. (Отмена некорректных сообщений) Удалить все сообщения вида (t, Par, i, l) для $t > time$ из $LS(i)$ и послать антисообщения вида $(t, Par, i, l)^*$ и сообщение $(t_i, O, i, 0)$.

Шаг 2. Если в буфере B_{ji} есть антисообщение $(time, Par, j, i)^*$, удалить его из буфера и сделать следующее:

2.1. Если $i = 0$ и $T_j > time$ то $T_i = time$.

2.2. Если в $LM(i)$ не найдено сообщение $(time, Par, j, i)$, то поместить принятое антисообщение в $L^*M(i)$.

2.3. Если в $LM(i)$ найдено сообщение $(time, Par, j, i)$, то удалить его из этого списка и отменить антисообщение. Если $t_j t_i$, то выход, иначе откат (см. 1.1.1. - 1.1.2.).

Шаг 3. Если $t_i < TMOD$, где $TMOD$ есть длина имитационного прогона, то

3.1. Если $i = 0$ и все $T_j > 0$ то определить $t\_ = minT_j$, все Ti сделать отрицательными и разослать всем остальным подмоделям сообщения вида $(t,0, i, j)$.

3.2. Если принятое сообщение имело вид $(t, 0, j, i)$, то удалить из всех списков элементы с меткой времени меньшей t.

3.3. Вычислить $X_i(t_i + 1)$ и сообщения $(t_i + 1, Par, i, j)$ для всех существующих дуг (i, j).
3.4. Увеличить t_i на единицу.
3.5. Если $t_i < TMOD$, то разослать все вычисленные сообщения и поместить их в $LS(i)$.

Глава 5

Лекции 11-12. Диагностика и предупреждение ошибок в имитационном моделировании

5.1 Графы событий

В настоящее время средства моделирования систем с дискретными событиями стали довольно обычным делом. Число их огромно, но нет никакого конца этому процессу из-за появления новых задач и областей исследования с одной стороны, и как новых аппаратных средств ЭВМ и технологий, так и новых языков программирования с другой. В этой ситуации имеет большое значение развитие технологии создания систем моделирования, которая уменьшила бы стоимость и трудоемкость этой работы. Имеются работы, которые посвящены этой проблеме, но они имеют дело главным образом с подходами к описанию систем [Buslenko – Concepcion] и с составом пакетов моделирования [21]. Однако, есть одна сторона проблемы, которая по некоторым причинам не достаточно освещена в литературе ([21] - один из немногих примеров) - отладка систем моделирования и имитационных моделей, хотя ясно, что этот класс программ имеет свои специфические особенности. В этой лекции мы будем обсуждать некоторых из наиболее обычных ошибок со следующих точек зрения: их происхождение, рекомендуемая реакция и место проверки.

Далее *календарь событий* и *список событий* – синонимы, поведение объекта в модели может быть представлено процессом или сопрограммой. В случае событийного моделирования мы будем использовать термины *обработчик события* или *процедура обработки события*.

5.2 Основные стадии моделирования и связанные с ними возможные ошибки

Основные этапы процесса имитационного моделирования описаны в лекции 1.

Хотя ошибки на первых двух шагах носят принципиальный характер и могут сделать напрасной всю последующую работу, не существует строгих методов их обнаружения, поскольку. вообще говоря, формулировка задачи в большой мере искусство.

На третьем шаге могут возникнуть ошибки, связанные с неверным выбором типов и диапазонов изменений переменных, опущением необходимых связей и, наоборот, перегрузкой модели несущественными деталями (последнее относится и к первым двум пунктам).

Ошибки на четвертом шаге весьма существенно сказываются на качестве эксперимента, но не на качестве самой разрабатываемой программной модели. Тоже можно отнести и к восьмому шагу.

На пятом шаге ошибки могут возникнуть как вследствие несоответствия формального описания модели возможностям выбранной системы программирования, так и вследствие "человеческого фактора недостаточной квалификации или невнимательности программиста. Именно эти ошибки возможно и должно определять автоматизированными средствами, (то же относится к шестому, седьмому и девятому шагам).

5.2.1 Ошибки в ходе моделирования

Различные подходы к моделированию имеют их определенные возможности создания ошибок. Ниже мы опишем некоторых из них.

Случай универсальных систем моделирования с императивным управлением

В ходе исполнения программы моделирования возможно, в общем случае, находить следующие:

1. Фатальные ошибки

   (a) Планирование события на прошедший момент времени. Это - одна из вопиющих ошибок, показывающая или неправильную логику моделирования взаимодействия процессов, или относительно неправильный алгоритм определения времени события.

   Рекомендация. Включать проверку этой ситуации в системе во время выполнения в телах процедур планирования. Диагностическое сообщение должно включить: имена намеченного события и планировщика (объект или процесс), системное время и, возможно, полный список событий (календарь событий).

   (b) Задержка на отрицательное время. Может возникать исключительно вследствие неправильного алгоритма вычисления времени задержки (может быть не непосредственно, но через неправильный алгоритм вычисления некоторых вовлеченных переменных).

   Рекомендация. Включать проверку этой ситуации в системе во время выполнения в теле задерживающейся процедуры. Диагностическое сообщение - как выше, за исключением имени намеченного события. Величина задержки также должна быть в сообщении.

   (c) Передача управления завершенному процессу. Свидетельствует об ошибках в логике взаимодействия моделирующих процессов. Завершение процесса означает, что соответствующий объект модели полностью выполнил свои функции и больше ей не принадлежит.

   Рекомендация. Как в пункте 1.a.

   (d) Передача управления текущему процессу или рекурсивный вызов процедуры обработки события. Может возникнуть при динамическом выборе обработчика события, возникающего в текущее время вследствие обработки текущего события. Свидетельствует о неверном алгоритме выбора.

   Рекомендация. Включать проверку этой ситуации в системе во время выполнения в телах операторов планирования. Диагностическое сообщение должно включить имена намеченного события и планировщика, системное время и, возможно, полный список событий.

2. Нефатальные ошибки

   (a) Планирование события, ранее уже запланированного. В принципе, возможно иметь в календаре событий несколько уведомлений об одном и том же событии, относящихся к разным моментам времени, например, о событии прихода требования на обслуживание. Однако это может означать и действительную ошибку вследствие планирования одного и того же уникального события разными объектами модели, что свидетельствует об ошибках в ее логике.

   Рекомендация. Как в пункте 1.a.

(b) Отмена незапланированного события. Возможно, не является ошибкой, если несколько объектов отменяют событие по принципу "кто раньше успеет но может свидетельствовать и об ошибках в алгоритме взаимодействия объектов.

Рекомендация. Включать проверку этой ситуации в системе во время выполнения в теле оператора отмены. Рекомендуемая реакция - игнорирование оператора и добавления сообщения к протоколу моделирования.

(c) Попытка планировать событие перед незапланированным событием или после него. Может указывать на ошибки в алгоритме взаимодействия объектов.

Рекомендация. Включать проверку этой ситуации в системе во время выполнения в телах процедур планирования. Рекомендуемая реакция - игнорирование оператора и добавления сообщения к протоколу моделирования.

(d) Попытка перепланирования незапланированного события. Может указывать на ошибки в алгоритме взаимодействия объектов.

Рекомендация. Включать проверку этой ситуации в системе во время выполнения в телах процедур перепланирования. Рекомендуемая реакция - выполнение как простого оператора планирования и добавление сообщения к протоколу моделирования.

3. Нежелательные динамические ситуации

(a) Длина календаря событий превышает данный контрольный уровень. Может указывать на неконтролируемый процесс размножения активных объектов. Приводит к переполнению памяти и резкому замедлению работы программы вследствие затрат на операции поиска по списку.

Рекомендация. Включать проверку этой ситуации в системе во время выполнения в теле процедуры управлением календаря события. Рекомендуемая реакция - добавление сообщения к протоколу моделирования.

(b) Число динамических объектов превышает данный контрольный уровень. Также может указывать на неконтролируемый процесс размножения объектов и сущностей. Приводит к переполнению памяти.

Рекомендация. Включать проверку этой ситуации в системе во время выполнения в теле процедуры управлением календарем событий. Хотя логически это было бы более удобно поместить проверку в генератор объектов, это трудно для реализации, потому что в многих пакетах моделирования для этой цели используются главным образом стандартные функции базовых языков программирования. Вместе с тем, запрос процедуры управлением календарем событий достаточно част. Рекомендуемая реакция - добавление сообщения к протоколу моделирования.

(c) Один из объектов удерживает управление в течение слишком долгого времени. Нет никакого продвижения в календаре событий. Может показывать на бесконечный цикл в процедуре обработки события. **Рекомендация**. Как в пункте выше.

В случае, когда мы используем некоторые специальные представления имитационных моделей, можно определить для них некоторые специфические ошибки.

5.3 Некоторые специальные случаи систем моделирования

5.3.1 Системы с оператором WAIT_UNTIL

Когда мы используем оператор "WAIT_UNTIL условие могут возникнуть некоторые специфические ситуации. Среди них:

1. Слишком долго ожидание. Процесс ожидает дольше указанного времени. Может быть, выполнение условия невозможно вообще.

Рекомендация. Чтобы избежать этой ситуации энергия срок должен быть добавлен к формату оператора WAIT_UNTIL. Если это не сделано, то как в пункте 3.б предыдущего раздела.

2. Все процессы находятся в состоянии ожидания. Ситуация, подобная смертельному объятию. В этом случае монитор событий найдет пустой календарь событий и, соответственно, остановит моделирование.

   Рекомендация. Чтобы предупредить экспериментатора относительно возможной ошибки в алгоритме, список ожидающих объектов (процессов) с их условиями ожидания должен быть добавлен к протоколу моделирования. Хотя мы обсуждаем ошибки во время выполнения, в этом разделе, кажется подходящим упомянуть специфическую ошибку, когда объект ожидает условия, описанного в переменных, которые никогда не изменяются (отсутствуют в левых частях операторов присваивания и не являются результатами вызовов процедур). Конечно, эта ошибка должна быть найдена в шаге компиляции, но не все компиляторы делают это.

5.3.2 Системы, ориентированные на сигналы

Посылка и получение сигнала - это специфические события, с которыми связан ряд ошибок и нежелательных ситуаций:

1. Нет ни одного приемника для отправителя. Объект производит сигнал, который никакие другие объекты не ожидают. Ситуация возможна в так называемых "широковещательных моделях когда не имеется никаких строгих связей между выходами и входами объектов, и сигналы посылаются и получаются от эфира. Эта ошибка не фатальна для отображения модели с динамически существующими приемниками.

   Рекомендация. Добавлять сообщение к протоколу моделирования.

2. Нет ни одного отправителя для приемника. Объект ожидает сигнал, который никакой другой объект не может производить. Эта ошибка подобна упомянутой в секции выше (случай бесконечного ожидания) и без сомнения является фатальной в случае, когда объекты не временные, но может не быть фатальные в противоположном случае, потому что отправитель может появляться в модели позже.

   Рекомендация. Добавляют сообщение к протоколу моделирования.

3. Полученный сигнал имеет неправильный тип. Приемник получает сигнал, но он имеет неправильный тип. Эта ситуация невозможна в случае кодирования правильных моделей со строгой связью входов/выходов, но может происходить в случае "широковещательных моделей когда, например, переменная сигнала описана как указатель на сигнал класса A, объект ожидает сигнала одного из подклассов A, а динамическое значение переменной - указатель на экземпляр одного из других подклассов A.

   Рекомендация. Используют стандартные опции проверки типов, базовой системы программирования. Когда ситуация возникает, добавлять сообщение к протоколу моделирования со значениями параметров полученного сигнала и, по выбору экспериментатора, остановить моделирование, или продолжить. Для перехвата управления от обработчика ошибок системы, использовать механизм обработки прерываний.

4. Полученный сигнал - "из прошлого". Ситуация - не ошибка в случае распределенного моделирования, но должна быть невозможна в противном случае (подобна планированию события в прошлом).

   Рекомендация. В случае нераспределенного моделирования – как в случае 1.а. раздела 1.3.

5.3.3 Транзактно-ориентированные системы моделирования

Специфические ошибки в транзактно-ориентированных системах моделирования связаны главным образом с неправильным использованием средств обслуживания и синхронизации потоков. Наиболее часты - следующие:

1. Транзакт поступает в блок освобождения ресурса прежде, чем этот, или другой транзакт был в соответствующем блоке занятия ресурса. Эта ошибка главным образом возникает тогда, когда число ресурсов рассчитывается динамически, вследствие неправильного алгоритма, но может возникать также в результате задания неправильной структуры блок-схемы модели. Все же некоторые модели могут позволять эту ситуацию.

   Рекомендация. Добавить, сообщение к протоколу моделирования и позволить транзакту идти к следующему блоку.

2. Нет пары для блока синхронизации (MATCH в GPSS, например). Ситуация невозможна в хороших GPSS-подобных системах со статическими блок-схемами и прямым указанием номера парного блока как константы, но может происходить в транзактно-ориентированных системах с динамическими блок-схемами (см., например, [23]) или если номер парного блока рассчитывается динамически.

   Рекомендация. В системах со строгими структурами блок-схемы - проверять эту ситуацию при анализе структуры модели на стадии предвыполнения, в системах с динамически изменяющимися структурами блок-схем - добавлять и удалить эти блоки только парами. Если ситуация происходит как результат вычисления номера парного блока, то остановка моделирования с добавлением сообщения к протоколу моделирования.

3. имеются "мертвые"ветви в блок-схеме модели. Имеются части блок-схемы модели, которые являются неспособными получить какой-либо транзакт. Ситуация может происходить или из-за ошибок в описании модели или кодировании в случае статически определенных путей транзактов, или из-за ошибок в алгоритмах вычисления этих путей в динамическом случае. В динамическом случае ситуация может происходить и как результат стохастического поведения модели следовательно, принципиально, ошибка не фатальная.

   Рекомендация. Если пути транзактов определены статически, ситуация может быть обнаружена при анализе структуры модели на стадии предвыполнения, в противном случае она может быть обнаружена только на стадии послеопытного анализа. Для последней цели статистика по использованию всех блоков должна быть в заключительном протоколе моделирования (как это присутствует в стандарте GPSS).

Литература

[1] Миков А.И. Моделирование вычислительных систем. Учебное пособие по спецкурсу. – Пермь: ПГУ, 1982.

[2] Прицкер А. Введение в имитационное моделирование и язык СЛАМ-2. – М.: Мир, 1987.

[3] Бусленко Н.П. Моделирование сложных систем. – М.: Наука, 1978.

[4] Молчан С.И., Преловская А.А., Родионов А.С. Комплекс программ генерации случайных процессов с заданными характеристиками //Системное моделирование в информатике (СМ-13) – Новосибирск, 1988. – С. 70–81.

[5] Rodionov A.S., Choo H., Youn H.Y. Process simulation using randomized Markov chain and truncated marginal distribution // Supercomputing. – 2002. – 1. – P. 69–85.

[6] Ермаков С.М., Михайлов Г.А. Статистическое моделирование. – М.: Наука, 1982.

[7] Соболь И.М. Численные методы Монте-Карло. – М.: Наука, 1973.

[8] Искусство программирования на ЭВМ. Т. 2. Получисленные алгоритмы. – М.: Наука, 1980.

[9] С.И. Молчан. Об одном подходе к моделированию случайных процессов с заданными характеристиками // Моделирование вычислительных систем и процессов.– Пермь,1986.-с.59-66

[10] Zeigler B.P. System-Theoretic Representation of Simulation Models // IIE Transactions, Vol. 16, No. 1, 1984, P.19–34.

[11] Киндлер Е. Языки моделирования. – М.: Энергоатомиздат, 1985.

[12] Андрианов А.Н., Бычков С.П., Хорошилов А.И. Программирование на языке Симула–67. – М: Наука, 1985.

[13] Vaucher J.G. A WAIT-UNTIL algorithm for general-purpouse simulation languages // Proc. of the 1973 Winter Simulation Conf. – Amsterdam: Noth-Holland P.C., 1973, P. 77-83.

[14] Vaucher J.G., Davey D. Acceleration du WAIT-UNTIL pour l'ordonnacement conditional en simulation // Teor. et techn. inform. Log. et mater. Act Congr. AFCET, Gif-sur-Yvett, 1978, V1 - Suresnes, 1978. P. 486-495.

[15] Knuth D.E., McNeley J.L. SOL — a symbolic language for general purpose system simulation // IEEE Trans. Elec. Comp. –1964. – Vol. 13, P. 401.

[16] Шрайбер Т.Дж. Моделирование на GPSS. – М.: Машиностроение, 1984.

[17] Родионов А.С. Реализация функций блока ADVANCE в классе ССМО языка СИМУЛА-67 // Системное моделирование–6. – Новосибирск, 1981. – С. 79–88.

[18] Bargodia R.L., Chandy K.M., Misra J. A Message-based approach to discrete-event simulation. // IEEE Trans. on soft. eng. vol. se-13, n.6, 1987, P. 654–665.

[19] Schruben L. Simulation modelling with event graphs. // Communication of the ACM, Vol. 26, N. 11, 1983, P. 957–963.

[20] Concepcion A.I., Zeigler B.P. DEVS-formalism: a framework for hierarchical model development. // IEEE trans. on soft. eng. vol.14, n.2, 1987, P. 228–241.

[21] Comfort J.C. Simulation model testing. // Proc. of the 20th annual simulation simp. 1987, P. 185–196.

[22] Waknis P., Sztipanovits J., Comfort J.C. Simulation model testing. // Proc. of the 20th annual simulation simp. 1987, P. 185–196.

[23] Родионов А.С. Некоторые вопросы применения языка СИМУЛА-67 для создания специализированного математического обеспечения // Системный анализ и исследование операций. - Новосибирск: ВЦ СО АН СССР, 1979. С. 23-30.