ВЫЧИСЛИТЕЛЬНАЯ ТЕХНИКА И ПРИКЛАДНАЯ МАТЕМАТИКА

УДК 510.5

А.В. Пруцков

ЛИНЕЙНЫЕ НОРМАЛЬНЫЕ АЛГОРИТМЫ

Предложена модификация нормальных алгоритмов Маркова, которая заключается в введении возможности последовательного выполнения подстановок. Данная модификация позволила уменьшить количество подстановок и трудоемкость решения классических задач теории нормальных алгоритмов Маркова.

***Ключевые слова**: алгоритмические модели, нормальные алгоритмы Маркова, обобщения нормальных алгоритмов.*

Введение. Задача обращения – задача с линейной трудоемкостью. Целью данной работы является разработка модификации нормальных алгоритмов Маркова с числом выполненных подстановок, линейно зависящим от длины слова, что позволит сократить трудоемкость решения задач. Здесь и далее под трудоемкостью в отношении нормальных алгоритмов Маркова и их модификаций понимается количество выполненных подстановок, необходимых для решения задачи.

Нормальные алгоритмы Маркова являются одной из алгоритмических моделей [1]. В теории нормальных алгоритмов Маркова рассматриваются две классические задачи: задача обращения и задача удвоения.

Введем обозначения: n – количество букв алфавита $Z=\{Z_1, Z_2, \ldots, Z_n\}$; m – длина обращаемой строки ($m > 0$).

Рассмотрим задачу обращения. Пусть в k ячейках находится m пронумерованных шариков, при этом $k \geq 2m$. Необходимо разместить их в обратном порядке. Для решения задачи необходимо взять предпоследний шарик и положить его за последним шариком (рисунок 1). Затем взять следующий шарик слева и поместить его в первую пустую ячейку справа от последнего шарика и т. д.

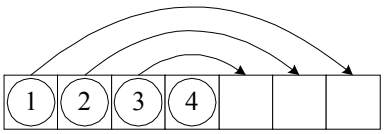


Рисунок 1 – Метод решения задачи обращения

Очевидно, что для решения задачи необходимо m – 1 операций поднятия и опускания шарика в ячейку, при этом последний шарик остается на месте. Трудоемкость описанного алгоритма является линейной.

Решим задачу обращения с помощью программы на алгоритмическом языке Паскаль. Программа включает следующие строки:

```
program p1;
const
  a: string='abc';
  b: string='';
var
  i,n: integer;
begin
  n:=length(a);
  for i:=1 to n do
    b:=b+a[n-i+1];
  writeln(b);
end.
```

Очевидно, что алгоритм обращения строки, лежащий в основе данной программы, также является линейным по трудоемкости.

Однако известные нормальные алгоритмы решения задачи обращения являются полиномиальными по трудоемкости. Данное обстоятельство можно объяснить недостаточностью средств, предоставляемых теорией нормальных алгоритмов Маркова.

Известные модификации нормальных алгоритмов Маркова. Рассмотрим некоторые известные модификации нормальных алгоритмов Маркова.

Н.М. Нагорный предложил обобщение нормальных алгоритмов Маркова (обобщенные нормальные алгоритмы Маркова, обобщения Н.М. Нагорного) [2]. Первое обобщение (тип σ) заключается в том, что алгоритм определяется конечным числом схем и на каждом шаге процесса вырабатывается указание того, какая схема должна выполняться на следующем шаге. У второго обобщения (тип σ') таких схем бесконечно много, и они порождаются некоторым нормальным алгоритмом. В третьем обобщении (тип σ'') бесконечны и сами схемы.

Подстановка обобщенного нормального алгоритма Маркова имеет следующий вид:

$$P_L \rightarrow P_R (I),$$

где P_L – левая часть подстановки; P_R – правая часть подстановки; I – индекс перехода к другой схеме.

Рассмотрим принцип работы схемы типа σ над словом R.

Алгоритм предписывает проверить, встречается ли одна из левых частей подстановок первой схемы системы в слове R. Если ни одна из левых частей не встречается в слове R, то алгоритм завершает работу. Если же левая часть одной из подстановок встречается в слове R, то данная подстановка применяется к слову. Если примененная подстановка имела индекс, равный нулю, то алгоритм завершает работу. Если же индекс примененной подстановки отличен от нуля, то происходит переход к схеме, указанной в индексе подстановки. Далее выполнение алгоритма происходит аналогично. Результатом работы алгоритма считается слово R после применения всех подстановок.

Ограничим описание обобщенных алгоритмов Маркова типом σ. Принципы функционирования алгоритмов типа σ' и σ'' в данной статье не приводятся.

Также в работе [2] показано, что для любого обобщения типов σ, σ' и σ'' может быть построен эквивалентный им нормальный алгоритм Маркова.

И.А. Цветков предложил самомодифицируемые нормальные алгоритмы [3]. Самомодифицируемые нормальные алгоритмы являются обобщением нормального алгоритма Маркова, а также предложенных И.А. Цветковым алгоритмов Маркова с самопополнением схемы, с самоудаляемыми подстановками, с одноразовыми пополнениями.

Самопополняемые алгоритмы Маркова могут иметь пополняемые подстановки, которые добавляются в схему во время выполнения нормального алгоритма. При пополнении в начало схемы (слева при записи схемы как слова) модификация называется самопополняемым слева алгоритмом Маркова. При пополнении в конец схемы (справа при записи схемы как слова) модификация называется самопополняемым справа алгоритмом Маркова. Каждая пополняющая подстановка при применении добавляет в начало схемы или в ее конец одну простую или одну заключительную подстановку. Любая подстановка (простая, заключительная, пополняющая) применяется неограниченное число раз. Количество пополнений не ограничено. Пополнения одноуровневые, то есть добавляется простая или заключительная, но не пополняющая подстановка.

В схеме нормального алгоритма с самоудаляемыми подстановками используются подстановки, которые удаляются из схемы автоматически после их первого применения. В схеме такого алгоритма есть простые, заключительные и пополняющие подстановки, применяемые неограниченное число раз, а также простые и заключительные подстановки однократного применения. Количество пополнений не ограничено. Пополнения одноуровневые.

В схеме алгоритма Маркова с одноразовыми пополнениями используются подстановки, которые однократно автоматически добавляют в конец схемы одну простую или одну заключительную подстановку. В схеме такого алгоритма простые, заключительные и пополняющие подстановки применяются неограниченное число раз, но пополняющая подстановка производит пополнение однократно. Пополнения одноуровневые.

На основе анализа перечисленных модификаций выдвинем два требования к предлагаемой модификации:

1) модификация должна состоять из одной схемы в отличие от обобщенных нормальных алгоритмов Маркова;

2) схема алгоритма не должна изменяться в отличие от самомодифицируемых алгоритмов Маркова.

Линейные нормальные алгоритмы. В алгоритмах существуют три типа вычислительных процессов [4]: линейный, разветвляющийся и циклический.

Из-за особенности функционирования в нормальных алгоритмах Маркова возможна реализация только двух типов вычислительных процессов: разветвляющегося и циклического.

Ветвление в нормальных алгоритмах Маркова реализуется с помощью кодирования ситуаций левыми частями подстановок. В зависимости от сложившейся ситуации во время

решения задачи выбирается и выполняется соответствующая данной ситуации подстановка.

Циклический вычислительный процесс в нормальных алгоритмах Маркова определяется, как правило, двумя подстановками. Первая подстановка определяет тело цикла – подстановку, выполняющуюся в цикле. Вторая подстановка в своей левой части содержит условие окончания цикла. Таким образом, первая подстановка выполняется до тех пор, пока не выполнится вторая подстановка.

Очевидно, что для решения линейных задач с линейной трудоемкостью необходима возможность реализации линейного вычислительного процесса.

Предлагается модификация нормальных алгоритмов Маркова, которая заключается в изменении структуры схемы и порядка выполнения подстановок. Данную модификацию назовем линейными нормальными алгоритмами. Схема линейного нормального алгоритма имеет следующий общий вид:

$$P_1:\ P_1^{(1)} \to Q_1^{(1)};\ P_2^{(1)} \to Q_2^{(1)};\ \ldots;\ P_{k_1}^{(1)} \to Q_{k_1}^{(1)}\ [\bullet]$$

$$P_2:\ P_1^{(2)} \to Q_1^{(2)};\ P_2^{(2)} \to Q_2^{(2)};\ \ldots;\ P_{k_2}^{(2)} \to Q_{k_2}^{(2)}\ [\bullet]$$

$$\ldots$$

$$P_d:\ P_1^{(d)} \to Q_1^{(d)};\ P_2^{(d)} \to Q_2^{(d)};\ \ldots;\ P_{k_d}^{(d)} \to Q_{k_d}^{(d)}\ [\bullet]$$

В схеме обозначения P только с нижним индексом и P и Q с верхним и нижним индексами являются подстроками; $k_1, k_2, \ldots, k_d = 1, 2, \ldots$.

Каждая строка схемы содержит метку P с нижним индексом и соответствующие данной метке одну или несколько подстановок.

Метка строки может не совпадать с левой частью первой подстановки в строке.

Алгоритм работает со строкой R следующим образом. Схема просматривается сверху вниз, и из схемы выбирается та строка схемы, метка P которой содержится в строке R. Если метка P найдена, то выполняются последовательно слева направо подстановки, соответствующие метке P, в строке R, то есть каждая подстановка выполняется только один раз. В следующий раз подстановка может быть выполнена только при следующем просмотре схемы. Таким образом, в линейных нормальных алгоритмах подстановки выполняются последовательно и при просмотре сверху вниз. Как и в нормальных алгоритмах Маркова, подстановка заключается в замене в строке R первого вхождения слева подстроки из левой части подстановки на подстроку из правой части подстановки. После того как все подстановки, соответствующие метке P, выполнены, схема просматривается вновь, начиная с первой подстановки.

Подстановки, помеченные пустым символом ∅, выполняются для любой строки R.

Алгоритм заканчивает выполнение в двух случаях:

1) ни одна метка P не встречается в строке R;

2) выполнена заключительная подстановка, заканчивающаяся символом «•».

Если алгоритм заканчивает свою работу (прекращает выполнение) со строкой R, то алгоритм считается применимым к строке R. Результатом работы алгоритма является строка R после применения всех подстановок.

Если алгоритм никогда не заканчивает свое выполнение (зацикливается) со строкой R, то алгоритм считается неприменимым к строке R и результат работы алгоритма неопределен.

Вернемся к задаче обращения. Запишем схему нормального алгоритма обращения с использованием аппарата линейных нормальных алгоритмов. Будем использовать только одну вспомогательную букву. Схема состоит из следующих подстановок.

1) $Z_i\lambda$: $Z_i\lambda \to \lambda;\ \to +Z_i$;i = 1, 2, …, n
2) λ: $\lambda \to \varnothing\ \bullet$
3) ∅: $\to +\lambda$

Указатель λ устанавливается в конец строки (подстановка 3). Каждый символ строки, который находится слева от указателя λ, удаляется, а затем добавляется в конец строки (подстановка 1). Таким образом, символы строки удаляются перед указателем λ и появляются справа от него в обратном порядке. Условием окончания работы алгоритма является отсутствие символов слева от указателя λ (подстановка 2). При выполнении данного условия указатель λ удаляется из строки.

Пример 1. Рассмотрим работу линейного нормального алгоритма обращения на примере числа 123 по шагам для алфавита Z = {1, 2, 3}. За один шаг может выполняться более одной подстановки.

0.		123
1.	3) ∅: $\to +\lambda$	12<u>3λ</u>
2-3.	1) 3λ: $3\lambda \to \lambda;\ \to +3$	1<u>2λ</u>3
4-5.	1) 2λ: $2\lambda \to \lambda;\ \to +2$	<u>1λ</u>32
6-7.	1) 1λ: $1\lambda \to \lambda;\ \to +1$	<u>λ</u>321
8.	2) λ: $\lambda \to \varnothing\ \bullet$	321

На последнем шаге алгоритма получено число 321 – обращение числа 123. □

Сравним данные о трудоемкости и о количестве подстановок предложенного алгоритма обращения, а также обращающего нормального алгоритма, предложенного А.А. Марковым и Н.М. Нагорным [1], и обращающего

самопополняемого слева алгоритма, предложенного И.А. Цветковым [3] (таблица 1).

Использование линейных нормальных алгоритмов позволило изменить тип алгоритма с полиномиального на линейный, сократив трудоемкость алгоритма, и уменьшить количество подстановок.

Таблица 1

Название алгоритма обращения	Трудоемкость	Количество подстановок алгоритма	Тип алгоритма
Нормальный алгоритм обращения А.А. Маркова и Н.М. Нагорного	$m^2/2 + 5m/2 + 1$	$8n^2 + 8n + 15$	Полиномиальный
Нормальный алгоритм обращения И.А. Цветкова	$m^2/2 + 3m/2 + 3$	$8n^2 + 25$	Полиномиальный
Линейный нормальный алгоритм обращения	$2m + 2$	$2n + 2$	Линейный

Сводимость нормальных алгоритмов Маркова к линейным нормальным алгоритмам. Схему любого нормального алгоритма Маркова можно привести к схеме линейного нормального алгоритма по следующим правилам:

- добавить метку к каждой подстановке, причем метка равна левой части подстановки;
- если левая часть подстановки пустая, то обозначить метку пустым символом ∅.

Таким образом, для нормальных алгоритмов Маркова каждой метке соответствует строка только с одной подстановкой.

Пример 2. Приведем нормальный алгоритм удвоения, предложенный в работе [5], к линейному нормальному алгоритму.

Исходный нормальный алгоритм удвоения слова алфавита Z = {a, b} включает следующие подстановки.

1) $\alpha a \to a\beta a\alpha$
2) $\alpha b \to b\beta b\alpha$
3) $\beta aa \to a\beta a$
4) $\beta ab \to b\beta a$
5) $\beta ba \to a\beta b$
6) $\beta bb \to b\beta b$
7) $\beta \to \varnothing$
8) $\alpha \to \varnothing \bullet$
9) $\to \alpha+$

Воспользуемся правилами приведения и получим следующие подстановки схемы линейного нормального алгоритма:

1) αa: $\alpha a \to a\beta a\alpha$
2) αb: $\alpha b \to b\beta b\alpha$
3) βaa: $\beta aa \to a\beta a$
4) βab: $\beta ab \to b\beta a$
5) βba: $\beta ba \to a\beta b$
6) βbb: $\beta bb \to b\beta b$
7) β: $\beta \to \varnothing$
8) α: $\alpha \to \varnothing \bullet$
9) $\varnothing$: $\to \alpha+$ □

Возможность сводимости алгоритмов объясняется тем, что линейный нормальный алгоритм позволяет выполнять подстановки в том же порядке при просмотре сверху вниз, что и нормальный алгоритм Маркова.

Сводимость схемы любого нормального алгоритма Маркова к схеме линейного нормального алгоритма влечет два следствия:

1) любую задачу, которую можно решить с помощью нормального алгоритма Маркова, можно решить и с помощью линейного нормального алгоритма;

2) трудоемкость любого линейного нормального алгоритма будет не хуже трудоемкости нормального алгоритма Маркова, но не наоборот.

Обратная сводимость от линейных нормальных алгоритмов к нормальным алгоритмам Маркова возможна при выполнении двух условий:

1) в каждой строке схемы находится только одна подстановка;

2) метка строки равна левой части подстановки строки.

Задача удвоения строки. Рассмотрим еще одну классическую задачу теории нормальных алгоритмов Маркова – задачу удвоения, которая заключается в преобразовании слова R в слово RR.

Задачу удвоения можно решить и с помощью предложенных линейных нормальных алгоритмов. Для удвоения элементов строки необходимо добавлять последовательно в конец строки копии элементов, имеющихся в строке (рисунок 2). Очевидно, что трудоемкость такого алгоритма будет линейной.

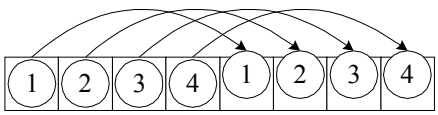


Рисунок 2 – Метод решения задачи удвоения

Схема линейного нормального алгоритма решения задачи удвоения слова алфавита Z = {a, b} будет состоять из следующих подстановок.

1) aα: aα → αa; β → βa
2) bα: bα → αb; β → βb
3) α: α → ∅; β → ∅ •
4) ∅: → +αβ

В конец строки добавляются указатели α и β (подстановка 4). Работа алгоритма заключается в перемещении указателя α влево и добавлении (копировании) символа перед указателем α в конец строки, обозначенный указателем β (подстановки 1-2). После того как перед указателем α не остается символов, алгоритм заканчивает выполнение (подстановка 3).

Пример 3. Рассмотрим работу линейного нормального алгоритма удвоения на примере той же строки bab по шагам.

0.		bab
1.	4) ∅: → +αβ	ba<u>b</u><u>α</u>β
2-3.	2) bα: bα → αb; β → βb	ba<u>α</u>bβb
4-5.	1) aα: aα → αa; β → βa	<u>bα</u>abβab
6-7.	2) bα: bα → αb; β → βb	<u>α</u>babβbab
8-9.	3) α: α → ∅; β → ∅ •	babbab

На девятом шаге алгоритма получена строка babbab – удвоение строки bab. □

Трудоемкость и количество подстановок предложенного линейного алгоритма удвоения, а также нормальных алгоритмов удвоения слова алфавита Z = {a, b}, предложенных в работах А.А. Маркова и Н.М. Нагорного [1] и В.А. Мощенского (см. пример 2) [5], представлены в таблице 2.

Таблица 2

Название алгоритма удвоения	Трудоемкость	Количество подстановок алгоритма	Тип алгоритма
Нормальный алгоритм удвоения А.А. Маркова и Н.М. Нагорного	$m^2/2 + 5m/2 + 2$	10	Полиномиальный
Нормальный алгоритм удвоения В.А. Мощенского	$m^2/2 + 3m/2 + 2$	9	Полиномиальный
Линейный нормальный алгоритм удвоения	$2m + 3$	7	Линейный

Использование линейных нормальных алгоритмов и в случае задачи удвоения позволило снизить трудоемкость алгоритма, сделав его линейным, а также уменьшить количество подстановок схемы.

Сводимость линейных нормальных алгоритмов к обобщенным нормальным алгоритмам Маркова. Любой линейный нормальный алгоритм можно свести к схеме типа σ обобщений Н.М. Нагорного. Сведение алгоритмов выполняется по следующим правилам.

1. В схему 1 выписываются первые подстановки каждой строки.

2. Если метке соответствует несколько подстановок, то каждая подстановка, кроме первой, записывается отдельной схемой, состоящей из одной подстановки. При этом схемы нумеруются соответствующими индексами, чтобы обеспечить их последовательное выполнение. Последняя подстановка, если она не является заключительной, имеет индекс 1, возвращающий управление схеме 1.

3. Если метка и левая часть первой подстановки не совпадают, то в схему 1 включается незначащая подстановка P → P, где P – метка. Первая подстановка и другие подстановки записываются в отдельные схемы по принципу, описанному в правиле 2. Незначащая подстановка передает управление схеме с первой подстановкой данной строки.

4. Заключительные подстановки имеют индексы 0.

Из данных правил следует, что количество схем обобщенного нормального алгоритма $N_{схем}$ зависит от числа строк схемы с более чем одной подстановкой и числа строк схемы, в которых метка отличается от левой части первой подстановки строки, и вычисляется по формуле:

$$N_{схем} = N_{подст} – N_{стр} + N_{мет} + 1,$$

где $N_{подст}$ – количество подстановок в строках схемы, в которых более одной подстановки; $N_{стр}$ – количество строк схемы, в которых более одной подстановки; $N_{мет}$ – количество строк схемы, в которых метка отличается от левой части первой подстановки строки; + 1 – основная схема (схема 1).

Пример 4. Сведем предложенный линейный нормальный алгоритм удвоения к обобщенному нормальному алгоритму Маркова. Воспользуемся правилами сведения и получим следующие схемы.

Схема 1:

1) aα → αa (2)
2) bα → αb (3)

3) $\alpha \rightarrow \varnothing$ (4)
4) $\rightarrow +\alpha\beta$ (1)
Схема 2:
1) $\beta \rightarrow \beta b$ (1)
Схема 3:
1) $\beta \rightarrow \beta a$ (1)
Схема 4:
1) $\beta \rightarrow \varnothing$ (0)

Рассчитаем количество подстановок по предложенной формуле. Исходные данные имеют следующие значения: $N_{подст} = 6$; $N_{стр} = 3$; $N_{мет} = 0$. Тогда расчет по формуле даст следующий результат:

$$N_{схем} = 6 – 3 + 0 + 1 = 4,$$

что равно полученному количеству схем. □

Как уже говорилось, в работе [2] показано, что для любого обобщенного нормального алгоритма может быть построен эквивалентный нормальный алгоритм Маркова. При этом любой линейный нормальный алгоритм можно свести к обобщенному нормальному алгоритму. Следовательно, для любого линейного нормального алгоритма может быть построен эквивалентный нормальный алгоритм Маркова.

В связи с возможностью данной сводимости возникает вопрос: означает ли сводимость линейных нормальных алгоритмов к обобщенным нормальным алгоритмам Маркова, что обобщения Н.М. Нагорного также являются и обобщением линейных нормальных алгоритмов? На этот вопрос невозможно однозначно дать положительный ответ, так как затруднительно найти ответ на следующий вопрос.

Вопрос заключается в следующем: можно ли свести любой обобщенный алгоритм Маркова к линейному нормальному алгоритму? Если ответ на второй вопрос положительный, то линейные и обобщенные нормальные алгоритмы являлись бы различными подходами к обобщению нормальных алгоритмов Маркова. Если ответ был бы отрицательным, то обобщения Н.М. Нагорного есть и обобщения линейных нормальных алгоритмов. Однако, чтобы дать ответ на второй вопрос, пришлось бы перебрать все возможные переходы управления между схемами обобщенных алгоритмов Маркова и представить их линейными нормальными алгоритмами, что является практически невыполнимой задачей.

С другой стороны, можно предложить многосхемную конструкцию, каждая схема которой является линейным нормальным алгоритмом, при этом данная многосхемная конструкция работает по тем же принципам, что и обобщения Н.М. Нагорного. В этом случае данная многосхемная конструкция будет обобщением обобщенных нормальных алгоритмов Маркова, а также нормальных алгоритмов Маркова и линейных нормальных алгоритмов.

Алгоритмы или алгорифмы. При использовании аппарата нормальных алгоритмов Маркова возникает вопрос об их правильном именовании: «нормальные алгориТмы» или «нормальные алгориФмы».

По всему ходу изложения в работе [1] используется термин «нормальные алгорифмы» как в отношении предложенной алгоритмической модели, так и к алгоритмам вообще.

Исходя из утверждений, приведенных на стр. 135, 143, 399, можно сделать вывод понятие «алгорифм» в работе [1] – синоним понятия «алгоритм». Однако «алгоритм» является более употребительным термином, поэтому в данной работе используется термин «нормальные алгоритмы».

Заключение. В данной статье проанализированы известные модификации и обобщения нормальных алгоритмов Маркова.

Предложена модификация нормальных алгоритмов Маркова, в которой введена возможность последовательного выполнения подстановок. Предложенная модификация названа линейными нормальными алгоритмами.

На примере классических задач теории нормальных алгоритмов Маркова показано, что предложенная модификация позволяет сделать алгоритмы более эффективными по сравнению с нормальными алгоритмами Маркова, уменьшая как число шагов, необходимых для решения задачи, так и количество подстановок в схеме.

Показано, что любой линейный нормальный алгоритм имеет эквивалентный ему нормальный алгоритм Маркова.

Прикладное значение предложенной модификации нормальных алгоритмов Маркова заключается в следующем:

1) линейные нормальные алгоритмы являются алгоритмической моделью, и, следовательно, могут использоваться для описания алгоритмов;

2) линейные нормальные алгоритмы могут использоваться для оценки сложности алгоритмов и сравнения их трудоемкости;

3) линейные нормальные алгоритмы – одна из простейших алгоритмических моделей, поэтому она может использоваться в учебных целях для обучения описанию решения задач в виде алгоритмов и определению их характеристик.

Библиографический список

1. *Марков А.А., Нагорный Н.М.* Теория алгорифмов. – М.: Наука, 1984. – 432 с.

2. Нагорный Н.М. Некоторые обобщения понятия нормального алгорифма // Тр. матем. ин-та АН СССР им. В.А. Стеклова, 52. – М.-Л.: Изд. АН СССР, 1958. – С. 66-74.

3. Цветков И.А. Обращающий самопополняемый слева алгорифм в алфавите с одной дополнительной буквой // Математическое и программное обеспечение вычислительных систем: Межвуз. сб. науч. тр. / Под ред. А.Н. Пылькина. – М.: Горячая линия-Телеком, 2008. – С. 4-9.

4. *Новичков В.С., Парфилова Н.И., Пылькин А.Н.* Алгоритмизация и программирование на Турбо Паскале: учеб. пособие. – М.: Горячая линия – Телеком, 2005. – 438 с.

5. *Мощенский В.А.* Лекции по математической логике. – Минск: Изд-во Белорус. ун-та, 1973. – 160 с.

УДК 681.3.06

А.И. Баранчиков, А.Ю. Громов

АЛГОРИТМ ГЕНЕРАЦИИ ФОРМАЛИЗОВАННОЙ МОДЕЛИ ПРЕДМЕТНОЙ ОБЛАСТИ

Предлагается алгоритм генерации формализованной модели гипотетической предметной области с заданными параметрами для проверки алгоритмов построения схем баз данных.

Ключевые слова: *предметная область, реляционные базы данных, модель, атрибут, функциональная зависимость.*

Цель. Разработка математического и алгоритмического обеспечения для программных средств тестирования алгоритмов проектирования схем реляционных баз данных.

Введение. При разработке алгоритмов для построения баз данных [1,2], кроме проверки их сходимости и расчёта временной сложности, возникает задача их статистического анализа путём экспериментальных исследований. Под экспериментальными исследованиями понимается многократное применение разработанных алгоритмов к различным наборам входных данных, как для проверки их корректности, так и для анализа эффективности, при изменяемых параметрах экспериментов.

Предлагается алгоритм генерации формализованной модели гипотетической предметной области с заданными параметрами.

Описание алгоритма.

Вход:

- количество l функциональных зависимостей $f \in F$;
- вероятностные характеристики моделируемых параметров.

Выход:

- множество атрибутов $A = (a_1, a_2, \ldots, a_n)$;
- схема отношения $R = (R_1, R_2, \ldots, R_k)$;
- множество функциональных зависимостей $F = (f_1, f_2, \ldots, f_m)$.

Теоретические исследования

Классификация параметров, характеризующих семантическую сложность формализованной предметной области. Основные параметры формализованных предметных областей, выступающие в качестве входных данных при проектировании реляционных баз данных, можно классифицировать следующим образом:

1) общее количество входных атрибутов $a \in A$.

От общего количества n входных атрибутов a зависит как временная сложность реализации алгоритма S, так и сложность набора входных функциональных зависимостей $f \in F$;

2) количество m функциональных зависимостей $f \in F$.

От данного параметра зависит количество шагов, необходимых для поиска транзитивных зависимостей, а значит, и на временную сложность реализации алгоритма S;

3) количество атрибутов $a \in A$ в каждой функциональной зависимости $f \in F$.

Данный параметр влияет на сложность функциональных зависимостей $f \in F$, также от него зависит временная сложность обработки каждой функциональной зависимости.

Варианты с разным количеством атрибутов в каждой функциональной зависимости $f \in F$:

1. $(B \rightarrow C, C \rightarrow DE, B \rightarrow F)$.
2. $(BH \rightarrow CEFL, C \rightarrow DSKY, BKX \rightarrow FNQPW)$.

4) глубина транзитивности входного набора функциональных зависимостей $f \in F$.

От глубины транзитивности зависит сложность связей между функциональными зависимостями $f \in F$, а значит, и временная сложность обработки этих связей. Моделирование данного параметра является отдельной задачей и в данной работе не рассматривается.

Варианты наборов функциональных зависимостей $f \in F$ с разной глубиной транзитивности:

1. ($B \to C$, $C \to D$).
2. ($B \to C$, $C \to D$, $D \to E$, $E \to F$, $F \to G$).

1) количество ключей в каждой функциональной зависимости:

1. ($B \to C$, $C \to D$).
2. ($BCD \to E$, $FG \to H$, $IJK \to ML$).

2) количество секретных атрибутов в каждой функциональной зависимости (секретные атрибуты выделены подчеркиванием):

1. ($BH \to C\underline{E}FL$, $C \to DSK\underline{Y}$, $B\underline{K}X \to FNQPW$).
2. ($B\underline{H} \to C\underline{EF}L$, $C \to \underline{DSKY}$, $B\underline{KX} \to \underline{FNQP}W$).

Далее будет рассмотрено моделирование следующих параметров:

- количество ключей n_K в каждой функциональной зависимости;
- количество открытых атрибутов n_A в каждой функциональной зависимости;
- количество секретных атрибутов n_S в каждой функциональной зависимости.

Вероятностная модель. В качестве инструмента моделирования рассмотренных ранее параметров использована вероятностная модель, основанная на нормальном законе распределения случайной величины. Нормальный закон распределения выбран на основании экспертных оценок, данных специалистами в области проектирования реляционных баз данных.

Для задания параметров предметной области используем следующие вероятностные характеристики:

μ - математическое ожидание конкретного параметра предметной области;

σ - среднеквадратическое отклонение параметра;

f(x) - функция плотности вероятности данного параметра.

Пусть *f(x)*имеет вид нормального распределения [3] (рисунок 1):

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{\left(-\frac{(x-\mu)^2}{2\sigma^2}\right)} \qquad (1)$$

Проквантуем данную функцию с шагом ***h***=0.5 (рисунок 2).

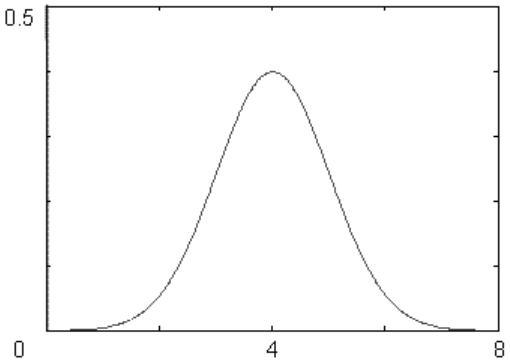


Рисунок 1 – Функция плотности вероятности (μ=4, σ=1)

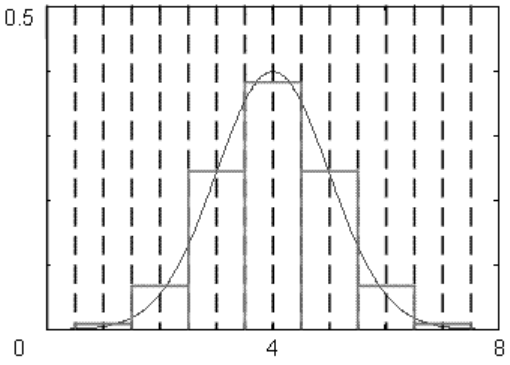


Рисунок 2 – Преобразование функции плотности вероятности

С учетом правила трёх сигм [3] практически все значения нормально распределённой случайной величины лежат в интервале [μ-3σ;μ+3σ], где μ - математическое ожидание случайной величины. Найдём вероятность событий *x*:

$$P(x) \approx \Delta x \cdot f'(x),$$
$$\Delta x = 2h,$$
$$f'(x) = (f(x - h) + f(x + h))/2,$$
$$P'(x) = 2h(f(x - h) + f(x + h))/2 =$$
$$= h\,(f(x - h) + f(x + h)). \qquad (2)$$

В силу того, что данная вероятностная модель необходима для нахождения величин целочисленных неотрицательных параметров, то на неё накладываются следующие ограничения:

1. Величина среднеквадратического отклонения σ выбирается из следующего условия:

$$3\sigma \le \mu - 1,$$

данное неравенство обеспечивает минимальное значение параметра, равное единице.

2. Величина задаваемого математического ожидания является целочисленной и не должна быть менее 2:

$$\mu \ge 2.$$

3. Количество величин рассчитываемых вероятностей зависит от величины среднеквадратического отклонения σ:

$$m = 6\sigma - 1.$$

В зависимости от заданных характеристик находим величины вероятностей $P'(x)$, где x – значение моделируемого параметра, а S_P — приблизительное значение площади фигуры под $f'(x)$.

Цель разработки: разработать алгоритм генерации формализованной модели предметной области для проектирования схемы реляционных баз данных.

В настоящее время большинство алгоритмов построения схем реляционных баз данных основано на приведении входных отношений к третьей нормальной форме (3НФ) (в редких случаях к нормальным формам более высокого порядка), то есть на преобразовании входной схемы $R = (R_1, R_2, \ldots, R_k)$ в схему $R' = (R'_1, R'_2, \ldots, R'_p)$, лишенную транзитивных зависимостей, которые в общем случае должна содержать генерируемая предметная область.

Под атрибутами понимается множество $A=(a_1, a_2, \ldots, a_n)$, которое впоследствии будут разделено на множество открытых X и секретных Y (таблица 1).

Таблица 1

Атрибут	Секретность	Ключевой
1	0	1
…		
n	0	0

Массив функциональных зависимостей заполняется в виде таблицы 2.

Таблица 2

Атрибут	Номер ФЗ	Левая часть	Правая часть
1	1	1	0
…	2	1	1
n	2	1	0

Экспериментальные исследования

Алгоритм.

Вход:

- количество l функциональных зависимостей $f \in F$;
- математические ожидания моделируемых параметров.

Выход:

- множество атрибутов $A = (a_1, a_2, \ldots, a_n)$;
- схема отношения $R = (R_1, R_2, \ldots, R_k)$;
- множество функциональных зависимостей $F = (f_1, f_2, \ldots, f_m)$.

Расчёт значений параметров x и вероятностей их появления $P'(x)$.

По введённым ранее ограничениям:

$$3\sigma \le \mu - 1 \rightarrow \sigma \le (\mu - 1)/3.$$

Наложим на σ ещё одно ограничение:

$$\sigma \le 1.$$

$P'(x) = h\cdot (f(x - h)+ f(x + h))$, где шаг квантования $h = 0.5$, то есть получаем:

$$P'(x) = 0.5\cdot (f(x - 0.5)+ f(x + 0.5)).$$

Значения x рассчитываются исходя из следующих выражений:

$$x_{max}=\mu + 3\sigma;\ x_{min}=\mu - 3\sigma.$$

Количество выборок m рассчитывается по формуле:

$$m = 6\sigma - 1.$$

В результате получаем набор значений x и приблизительную вероятность их появления:

$$x = [x_1; x_2; \ldots; x_m],$$

$$P' = [P'_1; P'_2; \ldots; P'_m].$$

Полученные данные будут использоваться в виде таблицы 3:

Таблица 3

Значение параметра x	Границы вероятности P' значения x	
	Левая граница	Правая граница
x_1	0	P'_1
x_2	P'_1	$P'_1 + P'_2$
...	...	...
x_m	P'_{m-1}	P'_m

С помощью таблицы 3 и генератора случайных чисел происходит выбор значений параметров предметной области.

Шаг 1. Расчёт вероятностей значений параметров предметной области и заполнение соответствующих массивов по образцу таблицы 3.

На данном шаге происходит заполнение таблиц вероятностей для следующих параметров предметной области:

- количество ключевых атрибутов в каждой функциональной зависимости $f \in F$**;**
- количество открытых атрибутов в каждой функциональной зависимости $f \in F$**;**
- количество секретных атрибутов в каждой функциональной зависимости $f \in F$**.**

Алгоритм *Mod* (μ);

вход: математическое ожидание μ моделируемого параметра;

выход: массив вероятностей P' значений x моделируемого параметра;

begin

σ = (μ - 1) / 3;

if σ > 1 **then** σ = 1;

$m = 2(\mu - 1) - 1$;
$x = \mu - 3\sigma$;
for $i = 1$ **to** m **do**
begin
if $x = \mu$ **then**
$Sp_i = 0.5(exp(-(x-\mu)^2/2\sigma^2)/2\sigma^2 + exp(-((x+0.5)-\mu)^2/2\sigma^2)/2\sigma^2)$;
else
$Sp_i = 0.5(exp(-((x-0.5)-\mu)^2/2\sigma^2)/2\sigma^2 + exp(-((x+0.5)-\mu)^2/2\sigma^2)/2\sigma^2)$;
$x = x+1$;
$Sa = Sa + Sp_i$;
end;
for $i = 1$ **to** m **do** $P_i = Sp_i/Sa$;
end;

Шаг 2. Заполнение массива функциональных зависимостей $f \in F$.

Расчёт количества открытых атрибутов n_A функциональных зависимостей $f \in F$ по заданному значению математического ожидания μ_A:

begin
$y = Random(100)$;
if $y \in [P'_{i-1}; P'_i]$ **then** $n_A = x_i$;
end;

Расчёт количества ключевых атрибутов $\boldsymbol{n}_K$ функциональных зависимостей $\mathbf{f} \in \mathbf{F}$ по заданному значению математического ожидания $\boldsymbol{\mu}_K$:

begin
$y = Random(100)$;
if $y \in [P'_{i-1}; P'_i]$ **then** $n_K = x_i$;
end;

Расчёт количества секретных атрибутов n_A функциональных зависимостей $f \in F$ по заданному значению математического ожидания μ_A:

begin
$y = Random(100)$;
if $y \in [P'_{i-1}; P'_i]$ **then** $n_S = x_i$;
end;

Каждый атрибут $a \in A$ может принадлежать как к левой, так и к правой части функциональной зависимости $f \in F$ либо сразу к обеим частям.

По полученным значениям параметров предметной области происходит построение множества функциональных зависимостей.

Алгоритм *Gen* (n_F, n_K, n_A, P', x);

вход:

- массив вероятностей P' значений x моделируемого параметра;
- количество открытых n_A, ключевых n_K и секретных n_S атрибутов;

выход:

- массив функциональных зависимостей f;
- массив атрибутов a;
- схема отношения R;

begin
$z = 1$;
for $i = 1$ **to** n_F **do**
begin
$y = Random(100)$;
if $y \in [P'_{i-1}; P'_i]$ **then** $n_A = x_{Ai}$;
$y = Random(100)$;
if $y \in [P'_{i-1}; P'_i]$ **then** $n_K = x_{Ki}$;
$y = Random(100)$;
if $y \in [P'_{i-1}; P'_i]$ **then** $n_S = x_{Si}$;
$n' = n_A + n_S$;
for j=1 **to** n_K **do**
begin
$K = a_z \cup K$;
$z = z+1$;
end;
for j=1 **to** n_S **do**
begin
$S = a_z \cup S$;
$z = z+1$;
end;
for j=1 **to** n_A **do**
begin
$A = a_z \cup A$;
$z = z+1$;
end;
end;
end;

где z – глобальный номер атрибутов, K – множество ключевых атрибутов функциональных зависимостей, A – множество открытых атрибутов, S – множество секретных атрибутов, *Random* – функция генерации случайных чисел.

В итоге получаем набор функциональных зависимостей, параметры которых удовлетворяют описанной ранее вероятностной модели.

Сходимость алгоритма. Предложенный алгоритм является сходящимся вследствие выполнения условий:

1. Количество n входных атрибутов $a \in A$ является ограниченным.
2. Так как множество входных атрибутов $A=(a_1, a_2, \ldots, a_n)$ ограниченно, то и количество шагов для обработки данного множества является конечным.

Расчёт временной сложности алгоритма. Под элементарной операцией будем понимать шаг алгоритма. Пусть n – количество входных атрибутов, тогда временную сложность алгоритма можно рассчитать следующим образом:

1) за $(2n + 4)$ шагов (худший случай) происходит расчёт вероятностей значений параметров предметной области, где n - количество входных атрибутов;

2) за l шагов происходит создание множества функциональных зависимостей. Каждый такой шаг содержит 6 n операций (худший случай), необходимых для расчёта значений параметров текущей функциональной зависимости и заполнения массива по полученным данным:

$$S = 6nl + 2n + 4.$$

Временная сложность алгоритма имеет порядок $O(nl)$.

Заключение. В качестве результатов проделанной работы можно выделить следующие:

- алгоритм генерации формальных предметные областей для проверки алгоритмов построения схем баз данных;
- доказана сходимость данного алгоритма;
- рассчитана временная сложность алгоритма.

Библиографический список

1. Баранчиков А. И., Громов А. Ю. Алгоритм построения схемы реляционной базы данных, содержащей атрибуты различной степени секретности // Информатика и прикладная математика. Рязанский государственный университет имени С.А. Есенина, 2008. С. 7 - 12.

2. Баранчиков А. И., Громов А. Ю. Алгоритм синтеза реляционной базы данных, учитывающий атрибуты различной степени секретности // Системы управления и информационные технологии, 2009. N3(37). С. 25 – 37.

3. Вентцель Е.С. Теория вероятностей. М.: Высшая школа; изд. 5-е, 1998. 576 с.

УДК 621.317.75:519.2

В.П. Корячко, Д.А. Перепелкин, А.И. Перепелкин

ПОВЫШЕНИЕ ЭФФЕКТИВНОСТИ ФУНКЦИОНИРОВАНИЯ КОРПОРАТИВНЫХ СЕТЕЙ ПРИ ДИНАМИЧЕСКИХ ИЗМЕНЕНИЯХ В ИХ СТРУКТУРЕ И НАГРУЗКАХ НА ЛИНИЯХ СВЯЗИ

Предложен алгоритм адаптивной ускоренной маршрутизации, повышающий эффективность функционирования корпоративных сетей.

Ключевые слова: *адаптивная ускоренная маршрутизация, динамические изменения, алгоритмы маршрутизации, корпоративные сети.*

Введение. Цель работы – разработка нового эффективного алгоритма поиска оптимальных маршрутов, повышающего эффективность функционирования корпоративных сетей. Характерной тенденцией развития современных корпоративных сетей является усложнение функций взаимодействия между их удаленными компонентами. Развитие новых сетевых технологий требует обеспечения качественного обслуживания современного трафика. Особую роль имеет эффективная маршрутизация пакетов данных в условиях всплеска трафика, локальных перегрузках и отказов отдельных элементов сети [1].

Ввиду сложности структур современных компьютерных сетей задача маршрутизации не решается в полной мере. В большинстве случаев это связано с маршрутизаторами, не справляющимися с поддержанием таблиц маршрутизации и выбором оптимальных маршрутов для данного класса трафика. Поэтому возникает задача исследования существующих алгоритмов маршрутизации с целью улучшения их характеристик или создания новых алгоритмов маршрутизации.

Теоретические исследования. В современных корпоративных сетях для продвижения пакетов по сети используют протоколы, выполняющие статическую и адаптивную (динамическую) маршрутизацию.

При статической маршрутизации все записи в таблице имеют неизменяемый, статический статус, что подразумевает бесконечный срок их жизни. Записи о маршрутах составляются и вводятся в память каждого маршрутизатора вручную администратором сети. При изменении состояния сети администратору необходимо срочно отразить эти изменения в соответствующих таблицах маршрутизации, иначе может произойти их рассогласование, и сеть будет работать некорректно.

При адаптивной маршрутизации все изменения конфигурации сети автоматически отражаются в таблицах маршрутизации благодаря протоколам маршрутизации. Эти протоколы собирают информацию о топологии связей в сети, что позволяет им оперативно обрабатывать все текущие изменения.

Протоколы адаптивной маршрутизации бывают распределенные и централизованные.

При распределенном подходе все маршрутизаторы сети находятся в равных условиях, они вычисляют маршруты и строят собственные таблицы маршрутизации, работая в тесной кооперации друг с другом, постоянно обмениваясь информацией о конфигурации сети. При централизованном подходе в сети существует один выделенный маршрутизатор, который собирает всю информацию о топологии и состоянии сети от других маршрутизаторов. На основании этих данных выделенный маршрутизатор строит таблицы маршрутизации для всех остальных маршрутизаторов сети, а затем распространяет их по сети, чтобы каждый маршрутизатор получил собственную таблицу и в дальнейшем самостоятельно принимал решение о продвижении каждого пакета [2].

Применяемые сегодня в IP-сетях протоколы маршрутизации относятся к адаптивным распределенным протоколам, которые, в свою очередь, делятся на две группы:

- дистанционно-векторные алгоритмы (Distance Vector Algorithms, DVA);
- алгоритмы состояния связей (Link State Algorithms, LSA).

В дистанционно-векторных алгоритмах (DVA) каждый маршрутизатор периодически и широковещательно рассылает по сети вектор, компонентами которого являются расстояния (измеренные в той или иной метрике) от данного маршрутизатора до известных ему сетей. Получив от некоторого соседа вектор расстояний (дистанций) до известных тому сетей, маршрутизатор наращивает компоненты вектора на величину расстояния от себя до данного соседа. В конце концов, каждый маршрутизатор узнает обо всех имеющихся сетях и о расстояниях до них. Выбор маршрутов к каждой сети осуществляется на основании наименьшего значения метрики. Дистанционно-векторные алгоритмы хорошо работают только в небольших сетях. В больших сетях они периодически засоряют линии связи интенсивным трафиком, к тому же изменения конфигурации не всегда корректно могут отрабатываться алгоритмом этого типа, так как маршрутизаторы не имеют точного представления о топологии связей в сети, а располагают только косвенной информацией – вектором расстояний. Дистанционно-векторные алгоритмы в своей работе используют протоколы RIP, RIP 2, IGRP и EIGRP.

Алгоритмы состояния связей (LSA) обеспечивают каждый маршрутизатор информацией, достаточной для построения точного графа связей сети. Все маршрутизаторы работают на основании одного и того же графа, что делает процесс маршрутизации более устойчивым к изменениям конфигурации. Выбор маршрута до необходимой сети осуществляется на основании наименьшего значения метрики. Служебный трафик, создаваемый протоколами LSA, гораздо менее интенсивный, чем у протоколов DVA. Протоколами, основанными на алгоритме состояния связей, являются протокол IS-IS и протокол OSPF.

Анализ современных алгоритмов маршрутизации в корпоративных сетях показывает, что дистанционно-векторные алгоритмы используют в своей работе алгоритм Беллмана – Форда, трудоемкость которого составляет $O(N^3)$, где N – число маршрутизаторов в корпоративной сети, а алгоритмы состояния связей базируются на алгоритме Дейкстры с трудоемкостью $O(N^2)$.

При изменении пропускной способности линий связи в представленных алгоритмах происходит полный перерасчет таблиц маршрутизации. С увеличением размера корпоративной сети трудоемкость этой операции растет полиномиально, что не может не сказаться на производительности всей сети в целом. Разработка новых, более эффективных алгоритмов адаптивной маршрутизации позволяет уменьшить трудоемкость построения таблиц маршрутизации в корпоративных сетях.

Разработка алгоритма. Для повышения эффективности функционирования корпоративных сетей предложен алгоритм адаптивной ускоренной маршрутизации, который позволяет уменьшить трудоемкость построения таблиц маршрутизации до величины $O(N)$ в условиях динамически изменяющейся структуры сети и нагрузок на линиях связи.

Представим корпоративную сеть в виде неориентированного взвешенного связного графа $G = (V,E,W)$, где V – множество вершин, $||V|| = N$, E – множество ребер, $||E|| = M$, W – множество весов ребер, показанного на рисунке.

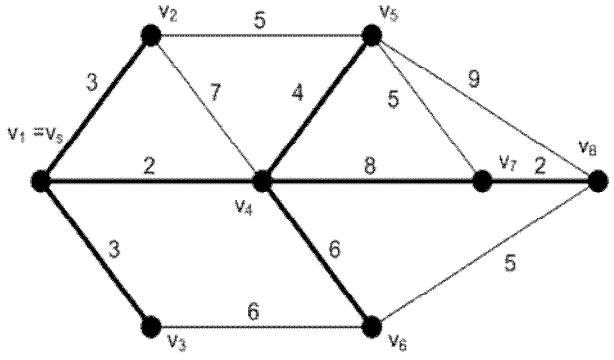


Рисунок – Граф G корпоративной сети

Пусть на графе G в некоторый момент времени уже решена задача поиска кратчайших путей до всех вершин множества $V_s=V\backslash v_s$ из начальной вершины v_s, т.е. построено дерево кратчайших путей с корнем в вершине v_s. Обозначим это дерево как T_g. На рисунке жирными линиями обозначено построенное дерево кратчайших путей.

Обозначим $w_{i,j}$ – вес ребра, соединяющего вершины v_i и v_j; $nw_{i,j}$ – новое значение веса, полученное в результате изменения значения метрики линии связи. Вершина v_j располагается ниже по иерархии дерева кратчайших путей относительно v_i. Обозначим E_T множество ребер, каждый элемент которого входит, по крайней мере, в один кратчайший путь из начальной вершины, E_R – множество остальных ребер. $E_R \cup E_T=E$, $E_R \cap E_T=\varnothing$. Обозначим V_T – множество вершин, до которых найден кратчайший путь из начальной вершины, V_R – множество остальных вершин. $V_R \cup V_T=V$, $V_R \cap V_T=\varnothing$.

Будем называть V_k – путем R_k совокупность подмножества $V^{(Vk)} \subseteq V$ вершин, через которые проходит кратчайший путь до вершины v_k из исходной вершины v_s, и подмножества $E^{(Vk)} \subseteq E$ ребер, составляющих этот путь.

Назовем V_k – деревом T_k совокупность подмножества $V_T^{(Vk)} \subseteq V$, состоящего из всех вершин, кратчайшие пути до которых из исходной вершины содержат вершину v_k, и подмножества $E_T^{(Vk)} \subseteq E$ ребер, составляющих эти пути после v_k при движении от вершины v_s.

Обозначим множество путей до вершины v_i из исходной вершины v_s через Π_i, где элемент множества $\pi_{i,k} \in \Pi_i$ будет множеством не повторяющихся ребер $e_{i,j} \in E$, образующих вместе путь, соединяющий v_s и v_i. Для всех $\pi_{i,k} \in \Pi_i$ поставим в соответствие некоторое число, равное сумме весов, входящих в него ребер, т.е. длину пути $d_{i,k} \in D_i$. На множестве Π_i задан селектор H, возвращающий кратчайший путь из множества Π_i. В том случае, если существуют несколько путей в Π_i с минимальной длиной, то выбирается один из них. Кратчайший путь до вершины v_i будем обозначать $\pi_i=H(\Pi_i)$, оценку длины $\pi_i - d_i$.

Для разработки алгоритма адаптивной ускоренной маршрутизации в корпоративных сетях сформулируем следующие теоремы.

Теорема 1. Если $nw_{i,j} > w_{i,j}$ и $e_{i,j} \in E_T$, то изменению могут подвергнуться кратчайшие пути и оценки их длин для вершин $V_T^{(Vj)}$.

Доказательство. Пусть увеличился вес ребра $e_{i,j} \in E_T$, которое входит, по крайней мере, в один кратчайший путь π_k, например в $\pi_{k,p}$. Вершины v_k, в кратчайшие пути до которых ребро $e_{i,j}$ не входит, будут составлять множество V_T вершин, кратчайшие пути до которых после изменения не изменятся (не изменится последовательность ребер и величины кратчайших путей). Действительно, пусть существует кратчайший путь $\pi_k = \pi_{k,p}$ до вершины v_k, и известно, что ребро $e_{i,j}$ не входит в этот путь. Тогда увеличение веса этого ребра со значения $w_{i,j}$ до $nw_{i,j}$ не изменит маршрута этого пути и не повлияет на его величину $d_{k,p}$, т.е. $\pi_{k,p}$ и $d_{k,p}$, поскольку еще до увеличения веса рассматриваемого ребра включение этого ребра в кратчайший путь приводило к увеличению длины пути. Все вершины, не вошедшие в множество V_T, будут составлять множество V_R. Кратчайшие пути до вершин множества $v \in V_R$ станут «недействительными», т.е. невозможно будет без дополнительного расчета сказать, останутся они такими же или кратчайший путь до них не будет включать изменившееся ребро. Теорема доказана.

Теорема 2. Если $nw_{i,j} < w_{i,j}$ и $e_{i,j} \in E_T$, то без изменения останутся кратчайшие пути для вершин множества $v \in V_T^{(Vj)} \cup V^{(Vi)}$, а для вершин множества $V^{(Vi)}$ неизменными останутся и оценки длин кратчайших путей.

Доказательство. Пусть уменьшился вес ребра $e_{i,j} \in E_T$, входящего в кратчайший путь $\pi_k = \pi_{k,p}$ до вершины $v_k \in V$. Ребро $e_{i,j}$ после изменения также будет входить в кратчайший путь π_k до вершины v_k. Поскольку вес ребра $w_{i,j}$ изменился, то измениться должны длины всех путей $\pi_{t,r}$, в которые входит это ребро. Действительно, если ребро $e_{i,j}$ входит в какой–либо кратчайший путь и вес этого ребра уменьшается, то это изменение не потребует изменения кратчайшего пути $\pi_{k,p}$ (последовательности ребер) и длина пути $d_{k,p}$ изменится (уменьшится) на величину изменения веса ребра. Пути π_s, $v_s \notin V_T^{(Vj)} \cup V^{(Vi)}$ станут «недействительными», т. е. невозможно будет без дополнительного расчета сказать, останутся они такими же или кратчайший путь до них будет включать изменившееся ребро. Теорема доказана.

Теорема 3. Если $nw_{i,j} > w_{i,j}$ и $e_{i,j} \notin E_T$, то исходное дерево кратчайших путей и оценки длин путей всех вершин не изменятся.

Доказательство. Пусть ребро, не входящее ни в один кратчайший путь, увеличивает свой вес $w_{i,j}$, $e_{i,j} \in E_R$. Никаких изменений дерева кратчайших путей при этом не происходит. Действительно, пусть ребро $e_{i,j} \in E$ входит в путь $\pi_{k,p}$ до некоторой вершины v_k, который не является кратчайшим для v_i, – $\pi_{k,p} \neq \pi_k$. Т.е.

существует такой путь $\pi_{k,t} = \pi_k$, что $d_{k,p} > d_{k,t}$. Тогда после увеличения веса $w_{i,j}$ увеличится оценка длины $d_{k,p}$ и неравенство $d_{k,p} > d_{k,t}$ останется справедливым. Т.е. кратчайший путь и его оценка до вершины v_k остается неизменной. Теорема доказана.

Теорема 4. Если $nw_{i,j} < w_{i,j}$ и $e_{i,j} \notin E_T$, то без изменения останутся кратчайшие пути и оценки их длин для вершин множества $V^{(Vi)}$.

Доказательство. Пусть уменьшился вес ребра $e_{i,j} \in E_R$, которое не входит ни в один кратчайший путь. Допустим, что это ребро входит в путь $\pi_{i,k} \neq \pi_i$ и $\pi_{j,p} \neq \pi_j$. Если изменившееся ребро $e_{i,j}$ не уменьшает оценок обеих инцидентных ему вершин v_i и v_j, т.е. $d_{i,k} \geq d_i$ и $d_{j,p} \geq d_j$, дерево кратчайших путей не изменится. Действительно, рассматриваемое ребро оказывает влияние, прежде всего на инцидентные ему вершины множества V. Если включение ребра $e_{i,j}$ в дерево не уменьшает оценок пути d_i, d_j, то такое включение только увеличит оценки вершин. Т.к. существовавшие до изменения пути до этих вершин имели меньшую длину, то данное ребро не включается в дерево кратчайших путей. Если включение этого ребра приводит к уменьшению оценки какой-либо из инцидентных вершин, например v_i, то эта оценка $d_{i,k}$ будет оценкой кратчайшего пути до вершины v_i, и ребро $e_{i,j}$ войдет в искомое дерево кратчайших путей. Это происходит в силу того, что после изменения не существует иного кратчайшего пути π_i до вершины v_i, кроме пути $\pi_{i,k}$, содержащего ребро $e_{i,j}$. Этот кратчайший путь $\pi_{i,k}$ не будет существовать, если не будет кратчайших путей до всех промежуточных вершин $v_p \in V^{(Vi)}$ этого пути. Кратчайшие пути до остальных вершин графа станут «недействительными», т. е. невозможно будет сказать, останутся они такими же или кратчайший путь до них будет включать изменившееся ребро. Теорема доказана.

Теорема 5. Если $nw_{i,j} > w_{i,j}$ и $e_{i,j} \in E_T$ и $nw_{i,j} > nw_{i,j}^{t}$ (точки вхождения в дерево), то изменению могут подвергнуться кратчайшие пути и оценки их длин для вершин $V_T^{(Vj)}$ и новые кратчайшие пути к этим вершинам будут проходить через ребра, состоящие в отношении парного перехода к ребрам этих вершин.

Доказательство. Пусть увеличился вес ребра $e_{i,j} \in E_T$, которое входит, по крайней мере, в один кратчайший путь π_k, например в $\pi_{k,p}$. Согласно теореме 1 вершины v_k, в кратчайшие пути до которых ребро $e_{i,j}$ не входит, будут составлять множество V_T вершин, кратчайшие пути до которых после изменения не изменятся (не изменится последовательность ребер и величины кратчайших путей). Для вершин $V_T^{(Vj)}$ среди парных переходов соответствующих этим вершинам будут находиться ребра, имеющие минимальную длину пути к этим вершинам. Теорема доказана.

Следствие. При увеличении веса ребра, входящего в дерево кратчайших путей для вершин $V_T^{(Vj)}$, маршрутная степень которых больше двух, новые кратчайшие пути будут проходить через ребра, состоящие в отношении парного перехода к ребрам, входящим в исходный граф.

Теорема 6. Если $nw_{i,j} < w_{i,j}$ и $e_{i,j} \notin E_T$ и новое значение $nw_{i,j} < nw_{i,j}^{t}$ (точки вхождения в дерево), то новые кратчайшие пути к вершинам множества $v \in V_T^{(Vj)} \cup V^{(Vi)}$ будут проходить через ребра, состоящие в отношении парного перехода к ребрам этих вершин.

Доказательство. Пусть уменьшился вес ребра $e_{i,j} \in E_R$, которое не входит ни в один кратчайший путь. Согласно с теоремой 4 без изменения останутся кратчайшие пути и оценки их длин для вершин множества $V^{(Vi)}$. Т.к $nw_{i,j} < nw_{i,j}$, то включение этого ребра приводит к уменьшению оценки какой-либо из инцидентных вершин, например v_i, и эта оценка $d_{i,k}$ будет оценкой кратчайшего пути до вершины v_i, и ребро $e_{i,j}$ войдет в искомое дерево кратчайших путей. Это происходит в силу того, что после изменения не существует иного кратчайшего пути π_i до вершины v_i, кроме пути $\pi_{i,k}$, содержащего ребро $e_{i,j}$. Этот кратчайший путь $\pi_{i,k}$ не будет существовать, если не будет кратчайших путей до всех промежуточных вершин $v_p \in V^{(Vi)}$ этого пути. Теорема доказана.

Следствие. При уменьшении веса ребра, не входящего в дерево кратчайших путей для вершин $V_T^{(Vj)} \cup V^{(Vi)}$, маршрутная степень которых больше двух, новые кратчайшие пути будут проходить через ребра, состоящие в отношении парного перехода к ребрам, входящим в исходный граф.

Использование доказанных выше теорем позволяет разработать алгоритм, уменьшающий размерность задачи поиска кратчайших путей в условиях, когда пропускная способность линий связи и структура сети динамически меняется.

Каждую вершину в алгоритме будет описывать структура данных VERTEX. Данная структура содержит индекс, равный индексу соответствующей вершины v_i; указатель на родительскую вершину $v_i.p$ (для корневой вершины он будет содержать null $v_i.p$=null), оценку длины оптимального маршрута $v_i.d$ и массив указателей на потомков по иерархии $v_i.c_j$

(для вершин–листьев он пустой). Полученное в процессе работы алгоритма дерево T будет представлять собой массив объектов типа VERTEX. Вершина v_i, i = 1..||V||, T.vs – начальная вершина. Каждое ребро графа G описывается структурой, которая определяется весом ребра e_i.w и указателями на инцидентные вершины e_i.v1, e_i.v2.

Для поиска оптимальных маршрутов будем применять алгоритм Дейкстры.

```
//Алгоритм Дейкстры. T – структура данных дерева оптимальных маршрутов, V – множество вершин графа, vs – начальная вершина.
DeicstraAlg(T,V,vs)
1.      Initialize(V,vs)
2.      Q.add(V,vs)
3.      Пока Q ≠ ∅
4.              vm = Extract-Min(Q);
5.              T.add(vm);
6.      Для всех инцидентных vm ребер vk
7.              Если d[vm.i]>d[vk.i]+w(vk,vm) то
8.              d[vm.i] = d[vk.i] + w(vk,vm);
9.                      p[vm.i] = vk;
                Конец Если
        Конец Для
        Конец Пока
Конец
```

В приведенной процедуре используется функция Extract – Min, возвращающая объект с минимальным ключом. Реализация данной функции и ее трудоемкость зависят от способа определения списка (очереди). Алгоритм вычисляет значения массивов d и p оценок длин оптимальных маршрутов и указателей на родительские таблицы соответственно. Полученные массивы необходимо сохранить.

```
// Процедура сохранения массивов
 SaveAr(v,i)
1.      Если i=0 то
2.              v.d1:=d;
3.              v.p1:=p;
        Иначе
4.              v.d2:=d;
5.              v.p2:=p;
        Конец Если
Конец
```

Для ребер, участвующих в построении оптимальных маршрутов, является целесообразным проводить расчет для инцидентных вершин, лежащих ниже по иерархии.

```
// Процедура нахождения деревьев оптимальных маршрутов для всех вершин графа
GetSPT(T,V,vs)
1.      Q.Add(vs)
2.      Пока Q≠∅
3.              v:=Q.Get
4.              Для всех v.ci
5.                      Q.Add(v.ci.v2)
                Конец Для
6.              Если v<>vs то
7.                      e:=E(v,v.p)
8.                      tw:=e.w;
9.                      e.w:=0;
10.                     DeicstraAlg(T,V,v)
11.                     SaveAr(v,0)
12.                     e.w:=M
13.                     DeicstraAlg(T,V,v)
14.                     SaveAr(v,1)
                Конец Если
        Конец Пока
Конец
```

В приведенной процедуре M – число, принятое за максимальное в данной системе. Процедура нахождения деревьев оптимальных маршрутов для графа обходит все вершины графа с формированием списков парных переходов для каждой вершины.

От дерева оптимальных маршрутов из заданной вершины легко перейти к списку парных переходов. Оценки длин оптимальных маршрутов в данном случае будут соответствовать приращениям потенциала этой вершины.

При поиске оптимального маршрута особой обработки, как было сказано выше, заслуживает вариант уменьшения веса ребра, не входящего в оптимальный маршрут. В этом случае необходимо предварительно проверить вершины, лежащие выше по иерархии.

```
// Процедура проверки на включение в поддерево вершины v вершин, лежащих выше по иерархии относительно вершины v
TestUp(v)
1.      v2:=v.P
2.      Пока v2.d>v.d+E(v,v2)
3.              v2.d:= v.d+E(v,v2)
4.              Для всех v2.Ci
5.                      Если v2.Ci<>v то
6.                      v2.Ci.d:=v2.d+E(v2,v2.Ci)
                        Конец Если
                Конец Для
7.              v:=v2;
8.              v2:=v2.P;
        Конец Пока
Конец
```

Для проверки срабатывания парного перехода используется условие целесообразности включения ребра в дерево. Отдельно обрабатываются случаи увеличения и уменьшения оценки длины оптимального маршрута.

```
// Процедура обработки изменения веса ребра будет выглядеть следующим образом:
PathPT(e,nw)
```

```
1.      v1=e.v1
2.      v2=e.v2
3.      Если v1.d>e.v2 то
4.              v1=e.v2
5.              v2=e.v1
        Конец Если
6.      Если nw>e.w то
7.              dt:=v2.d2
8.              pt:=v2.p2
9.              Если not e.InTree то
10.                     TestUp(v2)
                Конец Если
        Иначе
11.             dt:=v2.d
12.             pt:=v2.p
        Конец Если
13.     Q.Add(vs)
14.     Пока Q≠∅
15.             v:=Q.Get
16.             Для всех v.ci
17.                     Q.Add(v.ci.v2)
                Конец Для
18.             Если v<>vs то
19. Если v.p.d+E(v.p,v).w>pt[v.i].d+E(pt[v.i],v) то
20.             v.p:= pt[v.i]
21.             v.d:= pt[v.i].d+E(pt[v.i],v)
    Конец Если
                Конец Если
        Конец Пока
Конец
```

Рассмотрим работу алгоритма адаптивной ускоренной маршрутизации. Укрупненная схема алгоритма имеет следующий вид.

Шаг 1. Для вершины, являющейся листом дерева, производится поиск всех парных переходов без ограничений. Эти списки для удобства дальнейшей работы привязываются к вершине, инцидентной рассматриваемому ребру и расположенной ниже по иерархии.

Шаг 2. Если вершина не является листом дерева, то вычисляются парные переходы для этой вершины и выбираются лучшие значения потенциалов парных переходов для потомков вершины и собственных парных переходов. Подобная процедура выполняется для формирования списков парных переходов в случае увеличения и уменьшения веса ребра.

Шаг 3. Для каждой вершины формируется полный список парных переходов. Число элементов в каждом их этих списков не превышает количества вершин графа. Такое решение позволяет отказаться от предварительной сортировки потенциалов или приращений для парных переходов без значительного усложнения алгоритма обработки изменения.

Шаг 4. Для каждой вершины формируется полный список возможных маршрутов, проходящий через ребра, состоящие в отношении парного перехода, включая и ребра, входящие в дерево кратчайших путей.

Шаг 5. Анализируя полученную используемым протоколом маршрутизации информацию, определить, произошло ли изменение метрики для какого-либо ребра:

а) если да, то перейти к шагу 6;

б) иначе - к шагу 5.

Шаг 6. Используя список парных переходов, определить, требуется ли сделать парный переход:

а) если да, то перейти к шагу 7;

б) иначе – к шагу 9.

Шаг 7. Для каждой вершины, у которой в список возможных маршрутов входит ребро с изменившейся метрикой, определить путь минимальной длины и поместить его в дерево кратчайших путей, тем самым, построив новое дерево кратчайших путей на графе.

Шаг 8. а) передать пакеты по доступным эквивалентным маршрутам;

б) установить флаг передачи.

Шаг 9. Пересчитать точки вхождения в дерево и переформировать список маршрутов замены для каждой изменившейся вершины.

Шаг 10. Перейти к шагу 5.

Работа составных частей алгоритма основывается на использовании доказанных выше теорем, следовательно, можно сделать вывод о корректности работы всего алгоритма в целом.

Проведем анализ сложности алгоритма адаптивной ускоренной маршрутизации, для чего найдем верхнюю и нижнюю оценки его трудоемкости. Для этого определим оценки для процедур, составляющих алгоритм. Для всех процедур было найдено количество повторений каждого шага и стоимость для верхней и нижней оценок трудоемкости. Трудоемкость модифицированного алгоритма Дейкстры составляет:

$$\Omega(N^2 log_2 N).$$

Результаты анализа трудоемкости остальных процедур сведены в таблицах 1 – 3.

Таблица 1 – Трудоемкость процедуры SaveAr

№ шага п/п	$\Omega(N)$		$\Theta(N)$	
	Число повторений	Цена	Число повторений	Цена
1	1	1	1	1
2	1	1	0	1
3	1	1	0	1
4	0	1	1	1
5	0	1	1	1

Верхняя оценка трудоемкости процедуры SaveAr: $\Omega = 1 + 1 + 1 = 3$, а нижняя оценка:

$$\Theta = 1 + 1 + 1 = 3.$$

Таблица 2 – Трудоемкость процедуры GetSPT

№ шага п/п	$\Omega(N)$		$\Theta(N)$	
	Число повторений	Цена	Число повторений	Цена
1	1	1	1	1
2	N	1	N	1
3	N	1	N	1
4	N	1	N	1
5	N	1	N	1
6	N	1	N	1
7	N	1	N	1
8	N	1	N	1
9	N	1	N	1
10	N	$N\log_2 N$	N	$N\log_2 N$
11	N	3	N	3
12	N	1	N	1
13	N	$N\log_2 N$	N	$N\log_2 N$
14	N	3	N	3

Верхняя оценка трудоемкости процедуры GetSPT: $\Omega = 15N + 2N^2 log_2 N + 1$,

а нижняя оценка: $\Theta = 15N + 2N^2 log_2 N + 1$.

Переходя к асимптотическим выражениям, получаем верхнюю оценку трудоемкости построения дерева кратчайших путей и получения дополнительной информации:

$$\Omega(N^2 log_2 N)$$

и нижнюю оценку:

$$\Theta(N^2 log_2 N).$$

Тогда трудоемкость процедуры PathPT запишется следующим образом:

Верхняя оценка трудоемкости процедуры PathPT:

$$\Omega = 10 + 5N + 3(N - 1) + 6N + 2 = 14N + 9,$$

а нижняя оценка:

$$\Theta = 8 + 5N + N - 1 = 6N + 7$$

Таким образом, удается рассчитать дерево кратчайших путей за линейное время в худшем случаи. Такой результат удается получить путем использования дополнительной информации о парных переходах.

Переходя к асимптотическим выражениям, получаем верхнюю оценку трудоемкости:

$$\Omega(N)$$

и нижнюю оценку:

$$\Theta(N).$$

Таким образом, разработанный алгоритм адаптивной ускоренной маршрутизации позволяет повысить эффективность функционирования корпоративных сетей.

Таблица 3 – Трудоемкость процедуры PathPT

№ шага п/п	$\Omega(N)$		$\Theta(N)$	
	Число повторений	Цена	Число повторений	Цена
1	1	1	1	1
2	1	1	1	1
3	1	1	1	1
4	1	1	0	1
5	1	1	0	1
6	1	1	1	1
7	1	1	0	1
8	1	1	0	1
9	1	1	1	1
10	1	6N+2	0	1
11	0	1	1	1
12	0	1	1	1
13	1	1	1	1
14	N	1	N	1
15	N	1	N	1
16	N	1	N	1
17	N	1	N	1
18	N	1	N	1
19	N-1	1	N-1	1
20	N-1	1	0	1
21	N-1	1	0	1

Заключение. Разработка алгоритма адаптивной ускоренной маршрутизации позволяет повысить эффективность функционирования корпоративных сетей за счет уменьшения трудоемкости построения таблиц маршрутизации до величины порядка $O(N)$ в условиях динамически изменяющейся структуры корпоративной сети и характеристик линий связи.

Библиографический список

1. *Куракин Д.В.* Маршрутизаторы для глобальных телекоммуникационных сетей и реализуемые в них алгоритмы // Информационные технологии.– 1996. №2.

2. *Олифер В.Г., Олифер Н.А.* Основы компьютерных сетей – СПб.: Питер, 2009. – 352 с.

УДК 519.718

В.В. Тарасов, Н.В. Елкина

К ЗАДАЧЕ КОНТРОЛЯ ВХОДНОЙ ИНФОРМАЦИИ НА СХЕМАХ ИЗ ФУНКЦИОНАЛЬНЫХ ЭЛЕМЕНТОВ С ЗАДЕРЖКАМИ

Рассмотрена задача контроля входной информации на схемах из функциональных элементов в базисе $\{\&, \vee, \bar{}, z\}$, где z – тождественная функция с задержками.

***Ключевые слова**: функции алгебры логики с задержками, надежность и контроль, схемотехника.*

Введение. В исследованиях [1-5] по теории функций алгебры логики (ФАЛ) с задержками основной задачей являлось изучение методов синтеза вычислительных устройств на предмет быстродействия. ***Цель статьи***: рассмотрение задачи контроля входной информации на схемах из функциональных элементов в базисе $\{\&, \vee, \bar{}\}$ с добавлением к базису элемента, реализующего тождественную функцию z с задержками, дифференцированными по входу. Пусть t – момент времени подачи сигнала (нуля или единицы) на вход элемента z, тогда на выходе этот сигнал появляется в момент времени $t+\tau$, где $\tau=\tau_1$ при подаче на вход нуля и $\tau=\tau_2$ при подаче на вход единицы. Положим для определенности $\tau_1<\tau_2$. Под *ФАЛ с задержками* будем понимать пару функций $(f(x_1,...,x_n),\tau(x_1,...,x_n))$, где при одновременной подаче значений переменных $x_1=\sigma_1$, …, $x_n=\sigma_n$ на входы ФАЛ $f(x_1,...,x_n)$ на выходе получаем значение $f(\sigma_1,...,\sigma_n)$ с временной *внутренней* задержкой $\tau(\sigma_1,...,\sigma_n)\geq 0$, $\tau(\tilde{\sigma})\in R$. ФАЛ с задержками $(f(x_1,...,x_n), \tau(x_1,...,x_n))$ *существенно* зависит от переменной x_i, если существует пара соседних наборов по переменной x_i: $\tilde{\alpha}=(\sigma_1,...,\sigma_{i-1},0,\sigma_{i+1},...,\sigma_n)$ и $\tilde{\beta}=(\sigma_1,...,\sigma_{i-1},1,\sigma_{i+1},...,\sigma_n)$ такие, что $f(\tilde{\alpha})\neq f(\tilde{\beta})$ либо $\tau(\tilde{\alpha})\neq\tau(\tilde{\beta})$, в противном случае переменная x_i называется *фиктивной*.

При добавлении или изъятии фиктивных переменных у ФАЛ с задержками получившиеся функции не считаются равными. Причиной этого служит следующий важный пример. Рассмотрим схему $S(x,y)$ (см. рисунок 1), реализующую функцию x, где вход y является фиктивным с традиционно функциональной точки зрения.

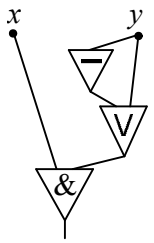


Рисунок 1

Если на входе y значение возникает на τ позже, чем значение на входе x, то на выходе схемы $S(x,y)$ значение также запаздывает на величину τ. Таким образом, фиктивные переменные могут принимать участие в управлении задержками.

Пусть $(f(x_1,...,x_n),0)$ – ФАЛ с нулевыми задержками. Тогда будем считать, что функция $(f(x_1(t+\tau_1),...,x_n(t+\tau_n)),0)$ имеет *внешнюю* задержку (или задержку на входе), равную $\max(\tau_1,...,\tau_n)-\min(\tau_1,...,\tau_n)$. Таким образом, в подсхеме некоторой схемы имеется задержка, которую можно рассматривать как сумму внутренней и внешней задержек (при этом внешняя задержка для подсхемы является составляющей для внутренней задержки всей схемы). Задержка для цепи в схеме в общем случае не является суммой внутренних задержек элементов цепи.

Приведем пример подсчета задержки цепи (см. рисунок 2), где τ_1, τ_2, τ_3 – внутренние задержки элементов цепи, а t, t_1, t_2, t_3 – моменты времени подачи сигнала. Тогда задержка цепи равна:

$$\tau_1+\left[\max(t_1,t+\tau_1)-\min(t_1,t+\tau_1)\right]+\tau_2+$$
$$+\left\{\max\left(\tau_1+\tau_2+\left[\max(t_1,t+\tau_1)-\min(t_1,t+\tau_1)\right],t_2,t_3\right)-\right.$$
$$\left.-\min\left(\tau_1+\tau_2+\left[\max(t_1,t+\tau_1)-\min(t_1,t+\tau_1)\right],t_2,t_3\right)\right\}+\tau_3$$

Задержка схемы есть максимум задержек по всем цепям. ▲

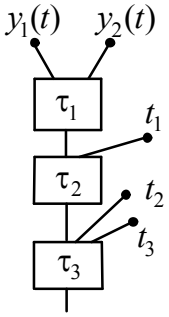
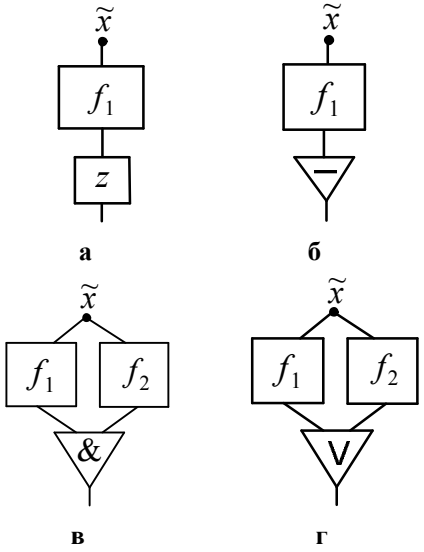


Рисунок 2

Основная часть. В базисе $\{\&, \vee, \bar{}, z\}$ подсчет задержек схемы может быть определен индуктивно по числу элементов в схеме.

1. Базис индукции. Задержки элементов $\&$, $\vee$, $\bar{}$ равны нулю. Тождественная функция с задержками $z = (x, \tau(x))$ имеет $\tau(0) = \tau_1 < \tau_2 = \tau(1)$.

2. Шаг индукции. Пусть определены две ФАЛ с задержками $(f_1(x_1,...,x_n), \tau'(x_1,...,x_n))$ и $(f_2(x_1,...,x_n), \tau''(x_1,...,x_n))$. Тогда для схем на рисунке 3 задержки соответственно равны: а) $\tau'(\tilde{x}) + \tau(f_1)$; б) $\tau'(\tilde{x})$; в)-г) $\max(\tau'(\tilde{x}), \tau''(\tilde{x}))$.



а б в г

Рисунок 3

Функция задержек схемы S_2 (см. рисунок 4), реализующей конъюнкцию двух переменных, определяется столбцом задержек $(2\tau_1, \tau_1 + \tau_2, \tau_2 + \tau_1, 2\tau_2)^T$ – соответственно входным набором $(0,0)$, $(0,1)$, $(1,0)$ и $(1,1)$. Функция задержек схемы S_3 (см. рисунок 5) равна: $3\tau_1$ на наборе $(0,0,0)$; $2\tau_1 + \tau_2$ на наборах $(0,0,1)$, $(0,1,0)$, $(1,0,0)$; $\tau_1 + 2\tau_2$ на наборах $(0,1,1)$, $(1,0,1)$, $(1,1,0)$; $3\tau_2$ на наборе $(1,1,1)$. Эта схема реализует конъюнкцию от трех переменных, причем на наборах с различным числом единиц схема на выходе выдает значения с разными задержками. Такое свойство сохранится при реализации конъюнкции от четырех переменных (см. рисунок 6): $4\tau_1$ на наборе $(0,0,0,0)$; $3\tau_1 + \tau_2$ на наборах с одной единицей; $2\tau_1 + 2\tau_2$ на наборах с двумя единицами; $\tau_1 + 3\tau_2$ на наборах с тремя единицами; $4\tau_2$ на наборе $(1,1,1,1)$.

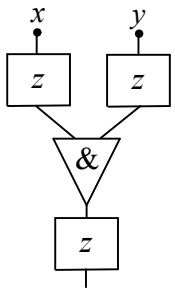


Рисунок 4
Схема S_2

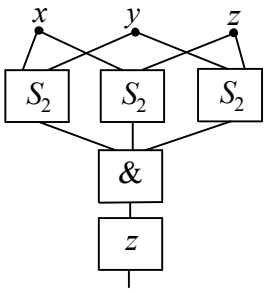


Рисунок 5
Схема S_3

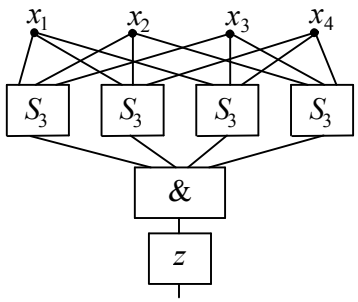


Рисунок 6
Схема S_4

Легко заметить, что функция задержек схемы S_n равна:

$$\begin{cases} n\tau_1 \text{ на наборе } (0,...,0), \\ (n-1)\tau_1 + \tau_2 \text{ на наборах с 1 единицей}, \\ (n-2)\tau_1 + 2\tau_2 \text{ на наборах с 2 единицами}, \\ \ldots\ldots\ldots\ldots\ldots\ldots \\ (n-k)\tau_1 + k\tau_2 \text{ на наборах с } k \text{ единицами}, \\ \ldots\ldots\ldots\ldots\ldots\ldots \\ n\tau_2 \text{ на наборе } (1,...,1). \end{cases} \quad (1)$$

В (1) все задержки различны. Действительно, пусть $k \neq m$. Тогда предположим, что $(n-k)\tau_1 + k\tau_2 = (n-m)\tau_1 + k\tau_2$. Отсюда получим $k(\tau_1 - \tau_2) = m(\tau_1 - \tau_2)$. Так как $\tau_1 \neq \tau_2$, то $k = m$, что противоречит предположению.

Рассмотрим теперь схему Σ с $2^n - 1 = m$ входами, реализующую конъюнкцию m-го

ранга, задержка в которой определяется числом единиц на входном наборе $(y_1,...,y_m)$ и равна $(m-r)\tau_1 + r\tau_2$, где r – число единиц в этом наборе. Множество входов схемы Σ разобьем на группы следующим образом: в первую группу попадают 2^{n-1} первых входов, во вторую группу – следующие 2^{n-2} входов и так далее, в последнюю группу попадает один вход. Итого имеем: $2^{n-1} + 2^{n-2} + ... + 1 = 2^n - 1$. В полученных группах произведем операцию отождествления входов. Полученная схема Σ' реализует конъюнкцию n-го ранга с задержками. Если на эту схему подать набор $(x_1,...,x_n)$ с k единицами, то на входах схемы Σ соответственно будет набор длины $2^n - 1 = m$ и с числом единиц, равным

$$2^{n-1}x_1 + 2^{n-2}x_2 + ... + 2^0 x_n, \qquad (2)$$

то есть десятичному числу, двоичная запись которого и есть набор $(x_1,...,x_n)$. Задержку схемы Σ' можно получить из формул (1) и (2):

$$\left(2^n - 1 - \sum_{r=1}^{n} 2^{n-r}x_r\right)\tau_1 + \sum_{r=1}^{n} 2^{n-r}x_r\tau_2 =$$
$$= 2^{n-1} + (\tau_2 - \tau_1)\sum_{r=1}^{n} 2^{n-r}x_r. \qquad (3)$$

Пусть имеются различные наборы $\tilde{x} \neq \tilde{y}$, имеющие равные задержки:

$$2^{n-1} + (\tau_2 - \tau_1)\sum_{r=1}^{n} 2^{n-r}x_r = 2^{n-1} + (\tau_2 - \tau_1)\sum_{r=1}^{n} 2^{n-r}y_r.$$

Отсюда имеем: $\sum_{r=1}^{n} 2^{n-r}x_r = \sum_{r=1}^{n} 2^{n-r}y_r$, что в двоичной записи означает, что $\tilde{x} = \tilde{y}$. Полученное противоречие доказывает, что при различных входных наборах схема реализует конъюнкцию n-го ранга с разными задержками.

В частности, при $\tau_1 = 0$ и $\tau_2 = 1$ из формулы (3) получим, что задержка на наборе $\tilde{x}$ равна (2), то есть десятичному числу, двоичная запись которого есть набор $\tilde{x}$.

Теперь построим *селекторное устройство с задержками* (СУЗ). Это будет тождественный булевый (n, n)-оператор, выдающий входной набор $\tilde{x}$ с задержкой, определяемой формулой (3). Для этого используем схему Σ' и $S(x,y)$ (рисунок 1), где на y-входы схем $S(x,y)$ подается выход схемы Σ' (см. рисунок 7).

Пусть $(\psi_1(\tilde{x}),...,\psi_n(\tilde{x}))$ есть булев (n,n)-оператор, реализуемый в базисе $\{\&, \vee, {}^-\}$ с нулевыми задержками. Тогда схема на рисунке 8 реализует этот оператор с дифференцированными задержками. По величине задержек однозначно может быть определена комбинация на входе.

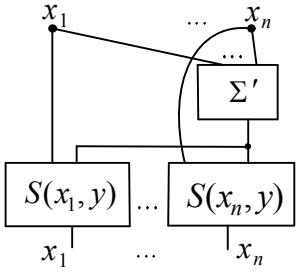


Рисунок 7

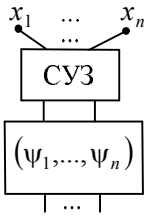


Рисунок 8

Результат, полученный в виде формулы (3), может быть получен при более общих начальных положениях. Пусть $(f(\tilde{x}), \tau(\tilde{x}))$ – произвольная булева функция с задержками, причем существуют два набора $\tilde{\alpha}$ и $\tilde{\beta}$ такие, что $\tau(\tilde{\alpha}) = \tau_1 < \tau_2 = \tau(\tilde{\beta})$. Составим матрицу:

$$\begin{pmatrix} x_1 & ... & x_i & ... & x_j & ... & x_k & ... & x_r & ... & x_n \\ \alpha_1 & ... & 0 & ... & 0 & ... & 1 & ... & 1 & ... & \alpha_n \\ \beta_1 & ... & 0 & ... & 1 & ... & 0 & ... & 1 & ... & \beta_n \end{pmatrix}.$$

Разобьем все переменные на группы: к группе N_1 отнесем переменные, под которыми в матрице стоит столбец $(0\,0)^T$, аналогично к группе N_2 – столбец $(0\,1)^T$, к группе N_3 – столбец $(1\,0)^T$, к группе N_4 – столбец $(1\,1)^T$, при этом некоторые переменные могут быть фиктивными. Осуществим подстановку в функцию $f(\tilde{x})$: на места переменных группы N_1 подставим константу 0, на места переменных группы N_2 подставим x, на места переменных группы N_3 подставим $\bar{x}$, на места переменных группы N_4 подставим 1. При склеивании всех входов получим функцию $\varphi(x)$, зависящую существенно от не более одного переменного функционально, но по задержке она существенно зависит от этого переменного, так как $\tau_\varphi(0) = \tau_1$, $\tau_\varphi(1) = \tau_2$. Рассмотрим четыре случая.

1. $\varphi(x)=0$. Тогда $\varphi(x)\vee x$ дает тождественную функцию x с задержками

$$\tau_x(0)=\tau_1,\ \tau_x(1)=\tau_2. \qquad (4)$$

2. $\varphi(x)=1$. Тогда $\varphi(x)\,\&\,x$ дает тождественную функцию с задержками (4).

3. $\varphi(x)=\bar{x}$. Тогда $\bar{\varphi}(x)$ дает тождественную функцию с задержками (4).

4. Наконец $\varphi(x)=x$ с задержками (4).

Таким образом, получаем следующий результат: для получения формул (3) достаточно иметь базис в P_2 с нулевыми задержками и произвольную функцию $f(\tilde{x})$ с задержками $\tau_f(\tilde{\alpha})=\tau_1<\tau_2=\tau_f(\tilde{\beta})$.

Замечание. Очевидно, что СУЗ можно использовать для контроля информации не только на входе схемы, но и в любом сечении схемы.

Дополнение. Приведем еще один пример применения задержки. Пусть имеется некоторая схема в базисе $\{\&,\vee,^{-}\}$ с дифференцированными задержками, но $\tau(0)=0$, $\tau(1)=1$. В этой схеме требуется проконтролировать работу некоторой подсхемы S_φ, реализующей функцию $\varphi(y_1,...,y_k)$. Для этого определим вспомогательную функцию:

$$\psi(y_1,...,y_k,y_{k+1})=\begin{cases}1, \text{ если } y_{k+1}=\bar{\varphi}(y_1,...,y_k),\\ 0, \text{ если } y_{k+1}=\varphi(y_1,...,y_k).\end{cases}$$

Ей отвечает схема S_ψ в базисе $\{\&,\vee,^{-}\}$ (без задержек).

Рассмотрим схему на рисунке 9.

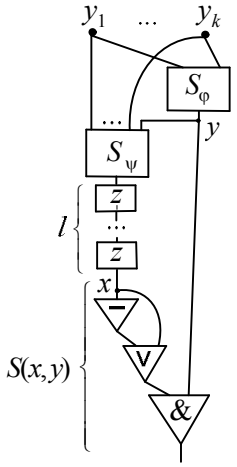


Рисунок 9

Если подсхема S_φ выдает на выходе ошибку $y=\bar{\varphi}(y_1,...,y_k)$, то подсхема S_ψ выдает на выходе 1, которое задерживается на l единиц. Концевая подсхема $S(x,y)$ выдает на выходе y с задержкой l. Если подсхема S_φ работает верно, то есть на ее выходе реализуется верное значение $y=\varphi(y_1,...,y_k)$, то подсхема S_ψ на выходе реализует значение 0, которое без задержек достигает входа x концевой подсхемы $S(x,y)$.

Выводы

1. Дан пример базиса с элементами дифференцированной задержки, в котором в виде макета построено селекторное устройство с такими дифференцированными задержками, что каждая входовая комбинация имеет свою, присущую только ей задержку. Селекторное устройство может быть подсоединено на вход любого булева оператора φ (без задержек) так, что новый булев оператор ψ имеет такие же дифференцированные задержки, как и селекторное устройство.

2. Описаны все базисы, состоящие из полных систем функций с нулевыми задержками и с добавлением функций с ненулевыми задержками, позволяющие построить селекторное устройство с дифференцированными задержками.

3. Приведен пример применения задержки для контроля работы выделенных подсхем данной схемы.

Библиографический список

1. *Бирюкова Л.А., Кудрявцев В.Б.* О полноте функций с задержками // Теория управляющих систем// Проблемы кибернетики, вып. 23. – М.: Наука. 1970. – С. 5-26.

2. *Лупанов О.Б.* О схемах из функциональных элементов с задержками// Проблемы кибернетики, вып. 23. – М.: Наука. 1970. – С. 43-82.

3. *Бирюкова Л.А.* Вопросы ℓ-полноты для функций с задержками // Проблемы кибернетики, вып. 31. – М.: Наука. 1976. – С. 53-76.

4. *Храпченко В.М.* Различие и сходство между глубиной и задержкой //Проблемы кибернетики, вып. 35. – М.: Наука. 1979. – С. 141-168.

5. *Ложкин С.А.* Поведение функции Шеннона для задержки схем из функциональных элементов в некоторых моделях //Тезисы докладов XII международной конференции «Проблемы теоретической кибернетики» (17-22 мая 1999 г.).

УДК 510

Г.С. Орлов

СЛУЧАЙНЫЕ ПРОЦЕССЫ НА БАЗЕ ОБОБЩЁННЫХ ДИНАМИЧЕСКИХ ВЕРОЯТНОСТНЫХ НАГРУЖЕННЫХ ГРАФОВ

Описывается процедура построения случайного процесса на базе обобщённых динамических вероятностных нагруженных графов. Анализируются её основные характеристики и свойства, даются определения сопутствующих понятий.

Ключевые слова: *динамический вероятностный нагруженный граф, случайный процесс, случайный граф, математическое моделирование.*

Введение. Цель работы – описать процедуру построения случайного процесса, в основе которого лежит новый математический объект – обобщённый динамический вероятностный нагруженный граф (ОДВНГ), рассмотреть её характерные особенности и ввести определения сопутствующих понятий.

Рассмотрим ОДВНГ [1] $G=(X_G,P_G)$, у которого $X_G=\{x_1,...,\ x_j,...,\ x_n\}$ - множество вершин, а $P_G=\{S^{(1)},...,S^{(j)},...,S^{(n)}\}$ - множество дуг и петель. Причём $S^{(j)}=\{\overline{U_{j1}},\overline{U_{j2}},...,\overline{U_{jk}},...,\overline{U_{jn}}\}$ есть множество дуг (петель при $k=j$), исходящих из вершины $x_j\in X_G$, а $\overline{U_{jk}}\overset{def}{=}\{U_{jk}^{(1)},U_{jk}^{(2)},...,U_{jk}^{(l_{jk})}\}$ - множество дуг из вершины $x_j\in X_G$ в вершину $x_k\in X_G$. В соответствии с определением ОДВНГ [1] будем считать, что в каждый момент времени $t\in T$ определены вероятности существования вершин графа $P_t(x_j)$ $\left(j=\overline{1,n}\right)$ и вероятности существования дуг $p_{jk}^{(l_i)}(t)$. Обозначим через $B(X_G)=(\varnothing,x_1,...,\{x_1,x_2\},...,X_G)$ булеан множества вершин X_G, все элементы которого упорядочены каким-либо способом (*то есть имеют порядковые номера*), соответственно через $B(\overline{U_{jk}})=(\varnothing,U_{jk}^{(1)},...,\{U_{jk}^{(1)},U_{jk}^{(2)}\},...,\overline{U_{jk}})$ - все возможные булеаны множеств $\overline{U_{jk}}$, также с упорядоченными элементами. Элементы конкретного булеана могут быть упорядочены любым способом, однако при этом они должны быть ранжированы по числу содержащихся в них элементов, то есть элемент булеана с меньшим числом элементов обязан иметь меньший порядковый номер, чем элемент булеана с большим числом элементов. Будем считать, что в каждом конкретном временном сечении может существовать один и только один элемент каждого такого булеана. Иначе говоря, если $\overline{B}=(B_1,...,B_j,...,B_m)$ ($B_j=(b_1^{(j)},b_2^{(j)},...,b_s^{(j)})$) - упорядоченная каким-либо способом совокупность всех определённых ранее булеанов, то для элементов $b_i^{(j)}$ каждого из них в любой момент $t\in T$ определен закон распределения вероятностей

$$B_j\begin{bmatrix} b_1^{(j)} & ... & b_i^{(j)} & ... & b_s^{(j)} \\ p(t)\left[p_t(b_1^{(j)}) & ... & p_t(b_i^{(j)}) & ... & p_t(b_s^{(j)})\right]\end{bmatrix},\ \left(\sum_{i=1}^{s}p_t(b_i^{(j)})=1\right),$$

ставящий в соответствие каждому элементу булеана вероятность его существования в данном временном сечении. Пусть $b_i^{(j)}=\{\tilde{b}_{ij}^{1},\tilde{b}_{ij}^{2},...,\tilde{b}_{ij}^{r}\}$ - произвольный элемент булеана B_j. Если все события вида: «*в данный момент* $t\in T$ *существует элемент* $\tilde{b}_{ij}^{k_1}$ *множества* $b_i^{(j)}$ *булеана* B_j» и «*в данный момент* $t\in T$ *существует элемент* $\tilde{b}_{ij}^{k_2}$ *этого же множества* $b_i^{(j)}$ *булеана* B_j» считать независимыми (*независимость в совокупности)*, то вероятность существования соответствующего элемента булеана (множества $b_i^{(j)}=\{\tilde{b}_{ij}^{1},\tilde{b}_{ij}^{2},...,\tilde{b}_{ij}^{r}\}$) будет равна $p_t(b_i^{(j)})=\prod_{k=1}^{r}p_t(\tilde{b}_{ij}^{k})$, где через $p_t(\tilde{b}_{ij}^{k})$ в данном случае обозначена вероятность существования элемента $\tilde{b}_{ij}^{k}$ (то есть – определённой

вершины или дуги ОДВНГ $G=(X_G,P_G)$) в момент времени $t\in T$. Таким образом, если в любом временном сечении $\tilde{t}\in T$ события, состоящие в существовании тех или иных вершин и дуг (в рамках одного булеана) графа $G=(X_G,P_G)$, считать независимыми в совокупности, то, зная величины $P_t(x_j)\left(j=\overline{1,n}\right)$ и $p_{jk}^{(l_i)}(t)$, мы можем найти вероятность существования любого элемента любого булеана в момент $\tilde{t}$. Для выполнения условия нормировки $\sum\limits_{i=1}^{s}p_t\left(b_i^{(j)}\right)=1$ перейдём к нормированным вероятностям существования элементов булеана $\tilde{p}_t\left(b_i^{(j)}\right)\overset{def.}{=}p_t\left(b_i^{(j)}\right)\Big/\sum\limits_{i=1}^{s}p_t\left(b_i^{(j)}\right)$.

Таким образом, будем считать, что в каждом временном сечении $t\in T$ определена многомерная случайная величина [2] $\bar{\xi}(t):\bar{B}\to\Re^m$, отображающая последовательность всех булеанов $\bar{B}$ в m–мерное пространство векторов $(\nu_1(t),\nu_2(t),...,\nu_m(t))$, где $\nu_j(t)\in\mathrm{N}$ - порядковый номер элемента $b_{\nu_j(t)}^{(j)}$ j-го булеана B_j, который (*один из всех элементов* B_j) существует в момент времени $t\in T$.

Очевидно, что упорядоченная последовательность номеров элементов булеана $(\nu_1(t),\nu_2(t),...,\nu_m(t))$ однозначно определяет для любого сечения $t\in T$ ОДВНГ $G(t)=(X_{G(t)},P_{G(t)})$, существующий в этом сечении. То есть в каждом временном сечении определена случайная функция [2] $\Xi_t(\omega):\bar{B}\to G(t)$, отображающая множество булеанов во множество ОДВНГ. Таким образом, получаем случайный процесс [2] $\Xi(\omega,t):\bar{B}\times T\to\bar{\bar{G}}$, где через $\bar{\bar{G}}$ обозначено множество всех подграфов ОДВНГ $G=(X_G,P_G)$.

Найдём основные числовые характеристики $\bar{\xi}(t):\bar{B}\to\Re^m$. Если ограничиться моментами порядка не выше второго, то совокупность случайных величин $(\nu_1(t),\nu_2(t),...,\nu_m(t))$ можно характеризовать вектором средних и ковариационной матрицей [2]. Очевидно, что $M\{\bar{\xi}(t)\}=(M\{\nu_1(t)\},M\{\nu_2(t)\},...,M\{\nu_j(t)\},...,M\{\nu_m(t)\})$.

Для каждого булеана определён закон распределения его элементов $\begin{matrix}B_j\\p(t)\end{matrix}\left[\begin{matrix}b_1^{(j)} & ... & b_i^{(j)} & ... & b_s^{(j)}\\ p_t\left(b_1^{(j)}\right) & ... & p_t\left(b_i^{(j)}\right) & ... & p_t\left(b_s^{(j)}\right)\end{matrix}\right]$, поэтому определён и закон распределения номеров данных элементов, то есть соответствие $\begin{matrix}N\\p(t)\end{matrix}\left[\begin{matrix}1 & ... & 2 & ... & s\\ p_t\left(b_1^{(j)}\right) & ... & p_t\left(b_i^{(j)}\right) & ... & p_t\left(b_s^{(j)}\right)\end{matrix}\right]$.

Следовательно, $M\{\nu_j(t)\}=\left[\sum\limits_{w=1}^{s}w\cdot p_t\left(b_w^{(j)}\right)\right]$, где символом $[\zeta]$ обозначена целая часть числа ζ. Ковариационная матрица для $\bar{\xi}(t)$ по определению [2] равна $K_{\bar{\xi}(t)}=\|k_{ij}(t)\|$ $\left(i,\ j=\overline{1,m}\right)$, где $k_{ij}(t)=COV(\nu_i(t),\nu_j(t))=$ $=M\{(\nu_i(t)-M\{\nu_i(t)\})\cdot(\nu_j(t)-M\{\nu_j(t)\})\}$. Соответствующая корреляционная матрица [2] будет $R_{\bar{\xi}(t)}=\|r_{ij}(t)\|$, где $r_{ij}(t)=\dfrac{COV(\nu_i(t),\nu_j(t))}{\sqrt{D\{\nu_i(t)\}\cdot D\{\nu_j(t)\}}}$, а $D\{\nu_j(t)\}=\sum\limits_{w=1}^{s}(w-M\{\nu_j(t)\})^2\cdot p_t\left(b_w^{(j)}\right)$.

Отметим характерные особенности случайного процесса $\Xi(\omega,t):\bar{B}\times T\to\bar{\bar{G}}$. Ясно, что в соответствии с вышеизложенным существование какой-либо дуги $U_{jk}^{(l)}$ в момент $\tilde{t}\in T$ не зависит от существования в этом же сечении $\tilde{t}$ инцидентных ей вершин x_j, x_r. Введём следующие определения.

Определение 1. ***Дугу*** $U_{jk}^{(l)}$ ***(петлю*** U_{kk}***) графа*** $G(t)=(X_{G(t)},P_{G(t)})$ ***будем называть замкнутой в сечении*** $\tilde{t}\in T$***, если в этом временном сечении существуют обе инцидентные этой дуге вершины*** x_j,x_k *(существует инцидентная петле вершина* x_k*).*

Определение 2. ***Дугу*** $U_{jk}^{(l)}$ ***(петлю*** U_{kk}***) графа*** $G(t)=(X_{G(t)},P_{G(t)})$ ***будем называть открытой в сечении*** $\tilde{t}\in T$***, если в этом временном сечении не существуют обе инцидентные этой дуге вершины*** x_j,x_k *(не существует инцидентная петле вершина* x_k*).*

Определение 3. ***Дугу*** $U_{jk}^{(l)}$ ***графа*** $G(t)=(X_{G(t)},P_{G(t)})$ ***будем называть полуотк-***

рытой слева в сечении $\tilde{t} \in T$***, если в этом временном сечении вершина*** x_j *(откуда исходит дуга)* ***не существует, а вершина*** x_k *(куда заходит дуга)* ***существует.***

Определение 4. ***Дугу*** $U_{jk}^{(l)}$ ***графа*** $G(t) = \left(X_{G(t)}, P_{G(t)}\right)$ ***будем называть полуоткрытой справа в сечении*** $\tilde{t} \in T$***, если в этом временном сечении вершина*** x_j *(откуда исходит дуга)* ***существует, а вершина*** x_k *(куда заходит дуга)* ***не существует.***

Определение 5. ***ОДВНГ будем называть замкнутым в сечении*** $\tilde{t} \in T$***, если в этом сечении все существующие дуги (петли) графа являются замкнутыми.***

Рассмотрим далее отдельное временное сечение $\tilde{t} \in T$. В момент времени $\tilde{t}$ определена многомерная случайная величина $\overline{\xi}(t) : \overline{B} \to \Re^m$, отображающая последовательность всех булеанов $\overline{B}$ в $m-$мерное пространство векторов $(\nu_1(t), \nu_2(t), ..., \nu_m(t))$, где $\nu_j(t) \in \mathrm{N}$ - порядковый номер элемента $b_{\nu_j(t)}^{(j)}$ $j-$го булеана B_j. Обозначим через $F_t(z_1, z_2, ..., z_m) \overset{def.}{=} P(\nu_1(t) < z_1, \nu_2(t) < z_2, ..., \nu_m(t) < z_m)$ $(z_j \in N)_{j=1}^m$ функцию распределения случайной величины $\overline{\xi}(t)$ в момент времени $t \in T$. Тот факт, что элементы всякого булеана упорядочены (*и ранжированы с учётом количества элементов в отдельном элементе булеана*), позволяет для каждого момента времени $\tilde{t}$ получить информацию о количестве вершин и дуг графа $G(\tilde{t}) = \left(X_{G(\tilde{t})}, P_{G(\tilde{t})}\right)$ по значениям $F_{\tilde{t}}(z_1, z_2, ..., z_m)$, вычисление которых не представляет затруднений. Действительно, из статистической независимости элементов булеанов следует, что $F_t(z_1, z_2, ..., z_n) = F_t(z_1) \cdot F_t(z_2) \cdot ... \cdot F_t(z_m)$. Рассмотрим любую из этих величин $F_t(z_j)$. Принципиально различаются два случая, когда булеан B_j есть булеан множества вершин графа $G = (X_G, P_G)$, и когда B_j - булеан какого-то множества дуг. Рассмотрим эти случаи по отдельности.

Если рассматривать булеан вершин $B_j = \left(b_1^{(j)} = \varnothing, b_2^{(j)} = x_1, ..., b_i^{(j)} = \left\{x_{i_1}, x_{i_2}, ..., x_{i_s}\right\}, ...\right)$, то процедура поиска значения $F_{\tilde{t}}(z_j) = P(\nu_j(\tilde{t}) < z_j)$ следующая. Выберем все элементы B_j с номерами, меньшими, чем z_j : $b_1^{(j)}, b_2^{(j)}, ..., b_{z_j-1}^{(j)}$. По формуле вероятности объединения несовместных событий найдём $P\left(\bigcup_{i=1}^{z_j-1} b_i^{(j)}\right) = \sum_{i=1}^{z_j-1} p_{\tilde{t}}\left(b_i^{(j)}\right)$. Вероятности $p_{\tilde{t}}\left(b_i^{(j)}\right)$ найдём по формуле вероятности произведения независимых событий. Предположим, что элемент булеана $b_{z_j-1}^{(j)}$ с наибольшим порядковым номером среди рассматриваемых содержит $\hat{n}(z_j - 1)$ элементов. Тогда значение $F_{\tilde{t}}(z_j) = P(\nu_j(\tilde{t}) < z_j)$ определяет вероятность события, состоящего в том, что "в сечении $\tilde{t}$ граф $G(\tilde{t}) = \left(X_{G(\tilde{t})}, P_{G(\tilde{t})}\right)$ будет иметь количество вершин, не большее чем $\hat{n}(z_j - 1)$".

Если анализируется булеан множества дуг $B_j = \left(b_1^{(j)} = \varnothing, b_2^{(j)} = U_{kr}^{(1)}, ..., b_i^{(j)} = \left\{U_{kr}^{(i_1)}, U_{kr}^{(i_2)}, ..., U_{kr}^{(i_s)}\right\}, ...\right)$, то для нахождения $F_{\tilde{t}}(z_j) = P(\nu_j(\tilde{t}) < z_j)$ поступаем аналогично. По формуле вероятности объединения несовместных событий найдём $P\left(\bigcup_{i=1}^{z_j-1} b_i^{(j)}\right) = \sum_{i=1}^{z_j-1} p_{\tilde{t}}\left(b_i^{(j)}\right)$. Вероятности $p_{\tilde{t}}\left(b_i^{(j)}\right)$ найдём по формуле вероятности произведения независимых событий. Предположим, что элемент булеана $b_{z_j-1}^{(j)}$ с наибольшим порядковым номером среди рассматриваемых содержит $\tilde{n}_{kr}(z_j - 1)$ элементов. Тогда значение $F_{\tilde{t}}(z_j) = P(\nu_j(\tilde{t}) < z_j)$ определяет вероятность события, состоящего в том, что "в сечении $\tilde{t}$ граф $G(\tilde{t}) = \left(X_{G(\tilde{t})}, P_{G(\tilde{t})}\right)$ будет иметь количество дуг, проведённых из вершины x_k в вершину x_r, не большее чем $n_{kr}(z_j - 1)$ ".

Очевидно, что значение функции $F_{\tilde{t}}(z_1, z_2, ..., z_n) = F_{\tilde{t}}(z_1) \cdot F_{\tilde{t}}(z_2) \cdot ... \cdot F_{\tilde{t}}(z_m)$ определяет вероятность следующих событий: "*В сечении* $\tilde{t} \in T$ *ОДВНГ* $G(\tilde{t}) = \left(X_{G(\tilde{t})}, P_{G(\tilde{t})}\right)$ *будет иметь вершин не более чем* $\hat{n}(z_j - 1)$", "*В сечении* $\tilde{t} \in T$ *ОДВНГ* $G(\tilde{t}) = \left(X_{G(\tilde{t})}, P_{G(\tilde{t})}\right)$

будет иметь дуг, проведённых из вершины x_k *в вершину* x_r*, не более чем* $n_{kr}(z_j-1)$", "*В сечении* $\tilde{t}\in T$ *ОДВНГ* $G(\tilde{t})=(X_{G(\tilde{t})},P_{G(\tilde{t})})$ *будет иметь дуг, инцидентных вершинам* x_k, x_r*, не более чем* $n_{kr}(z_j-1)+n_{rk}(z_j-1)$", "В сечении $\tilde{t}\in T$ ОДВНГ $G(\tilde{t})=(X_{G(\tilde{t})},P_{G(\tilde{t})})$ будет иметь дуг и петель не более чем $\sum_{k=1}^{n}\sum_{r=1}^{n} n_{kr}(z_j-1)$".

В реальных задачах моделирования случайных процессов на базе ОДВНГ принципиально важным является знание того, каким образом в данном временном сечении закон распределения вершин графа влияет на распределение множеств дуг (петель). Пусть $\nu_1(t)$ - случайная функция, определяющая для каждого сечения $t\in T$ номер элемента булеана множества вершин ОДВНГ (то есть указывающая те вершины $G(t)=(X_{G(t)},P_{G(t)})$, которые существуют в момент t). Соответственно пусть $\nu_j(t)$ - случайная величина, определяющая, какие именно дуги из множества $\overline{U_{jk}}=\left\{U_{jk}^{(1)},U_{jk}^{(2)},...,U_{jk}^{(l_{jk})}\right\}$ существуют в сечении t. Рассмотрим вероятность события $\left\{(\nu_1(t)=n_1)\cap(k_j^1\le\nu_j(t)\le k_j^2)\right\}$, $(n_1,k_j^1,k_j^2\in N;\ k_j^1<k_j^2)$. По теореме умножения имеем $P\left\{(\nu_1(t)=n_1)\cap(k_j^1\le\nu_j(t)\le k_j^2)\right\}=$ $=P\{\nu_1(t)=n_1\}\cdot P\left\{k_j^1\le\nu_j(t)\le k_j^2\mid\nu_1(t)=n_1\right\}$. То есть $P\left\{k_j^1\le\nu_j(t)\le k_j^2\mid\nu_1(t)=n_1\right\}=$ $=\dfrac{P\left\{(\nu_1(t)=n_1)\cap(k_j^1\le\nu_j(t)\le k_j^2)\right\}}{P\{\nu_1(t)=n_1\}}$.

Задача нахождения $P\{\nu_1(t)=n_1\}$ была решена ранее. Найдём $P\left\{(\nu_1(t)=n_1)\cap(k_j^1\le\nu_j(t)\le k_j^2)\right\}$.

В простейшем случае, когда события, состоящие в существовании конкретных множеств вершин и дуг графа (*в конкретном временном сечении*), являются независимыми $P\left\{(\nu_1(t)=n_1)\cap(k_j^1\le\nu_j(t)\le k_j^2)\right\}=$ $=P\{(\nu_1(t)=n_1)\}\cdot P\left\{(k_j^1\le\nu_j(t)\le k_j^2)\right\}$ и задача решена. Пусть теперь события $(\nu_1(t)=n_1)$, $(k_j^1\le\nu_j(t)\le k_j^2)$ - зависимы, тогда $P\left\{(\nu_1(t)=n_1)\cap(k_j^1\le\nu_j(t)\le k_j^2)\right\}=$ $=P\left\{(\nu_1(t)=n_1)\cap\left[\bigcup_{s=k_j^1}^{k_j^2}(\nu_j(t)=s)\right]\right\}=$ $=\left|\begin{array}{c}\text{учитывая несовместность}\\ \text{событий}\ (\nu_j(t)=s)\left(s=\overline{k_j^1,k_j^2}\right)\end{array}\right|=$ $=P\left\{\bigcup_{s=k_j^1}^{k_j^2}\left[(\nu_1(t)=n_1)\cap(\nu_j(t)=s)\right]\right\}=$ $=\sum_{s=k_j^1}^{k_j^2}P\left\{(\nu_1(t)=n_1)\cap(\nu_j(t)=s)\right\}$.

Кажется очевидным, что вероятности событий $(\nu_1(t)=n_1)\cap(\nu_j(t)=s)$ $\left(s=\overline{k_j^1,k_j^2}\right)$ зависят от совокупности конкретных вершин и дуг, входящих во множество вершин (элемент булеана вершин с порядковым номером n_1) и множество дуг (элемент булеана дуг с порядковым номером s), и поэтому определяются (задаются) применительно к конкретной моделируемой ситуации.

Заключение. В данной статье предлагается и анализируется алгоритм построения модели случайного процесса на базе введённого ранее автором [1] нового объекта – обобщённого динамического вероятностного нагруженного графа. Даются определения основных сопутствующих понятий, анализируются особенности и характерные свойства процесса.

Библиографический список

1. *Орлов Г.С.* Динамические вероятностные нагруженные графы. Определения, свойства, области применения. – Вестник РГРТУ. № 1 (выпуск 31). Рязань, 2010. – С. 48 – 57.

2. *Гихман И.И., Скороход А.В.* Введение в теорию случайных процессов. – М.: Из–во «Наука», 1965. – 655 с.