УДК 004.946

*ШИРОЧИН В.П.,*
*ШИЛОВ Ю.Н.*

# МИНИМИЗАЦИЯ ОБЪЕМА УПРАВЛЯЮЩЕГО СПИСКА ДЛЯ ПЛАНИРОВАНИЯ СОБЫТИЙ В PETRI-NETS МОДЕЛЯХ

Стохастическая сеть Петри является моделью реальных параллельных систем. Анализ процессов, происходящих в таких системах, требует большого количества времени. Минимизация управляющего списка в процессе моделирования работы системы позволяет сократить требуемое для анализа время. Системы моделирования, использующие оптимизированные управляющие списки, смогут давать более точные результаты моделирования за кратчайшие сроки.

Stochastic Petri net is a model of real parallel systems. Analysis of the processes occurring in such systems requires a large amount of time. Minimizing the control list in the simulation of the system reduces the required time for analysis. Modeling systems that use optimized control lists, will give a more accurate simulation result for the shortest possible time.

## Симуляция работы сети Петри

Управляющий список является ключевым элементом для реализации режима симуляции сетей Петри. Он содержит информацию обо всех осуществленных переходах и новой маркировке на каждом шаге, получаемой путем решения матричных уравнений сети. По списку собираются статистические данные о работе сети, таким образом, можно обнаружить её слабые и сильные стороны. Статистически значимый результат появляется при количестве событий симуляции более 10 000, что напрямую влияет на время обработки списка, соответственно задача минимизации его объема является крайней важной для получения быстрых и в высокой степени точных результатов анализа сети.

Управляющий список является таблицей данных о произошедших событиях во время моделирования работы сети. Для наиболее полного анализа сети необходимо фиксировать время события, тип события, задание и завершающую маркировку (после обработки события).

Наиболее известными редакторами сетей Петри, использующимися сейчас, являются PIPE 2.0, WoPeD и инфраструктура PetriNet-Kernel. Несмотря на широкий выбор возможностей, их объединяет отсутствие четкого и структурированного управляющего списка. Это, во-первых, скрывает от пользователя важные данные о реальных процессах, происходящих в сети, а, во-вторых, предоставляет систему «как есть», без возможности оптимизации и ускорения ее работы. В данной работе предложен редактор для построения и анализа управляющего списка с минимизацией времени его обработки.

## Проблемы замедления анализа сети и способы их решения

Для уменьшения времени анализа работы сети необходимо выяснить основные причины замедления данного процесса:

1) Необходимость сортировки списка по времени.
2) Увеличение времени выборки с увеличением списка.
3) Сложность отображение большого объема данных в графическом интерфейсе пользователя.

Варианты решения проблем:

1) Оптимальным способом сортировки в данном случае является сортировка слиянием. Но на практике сортировка в данном алгоритме симуляции работы сети не является наиболее необходимым этапом для реализации, так как большая часть управляющего списка уже изначально может быть отсортирована, таким образом дополнительная сортировка не нужна. Поэтому список можно разбить на части, проверить каждую из них на правильный порядок элементов, а затем объединить. При большом размере списка сложность этого метода будет целиком зависеть от скорости обхода всего массива событий.

2) В решении предыдущей проблемы сортировки списка уже была упомянута нецелесообразность этого этапа, причиной этому послу-

жила возможность вставки элемента на нужное место в списки во время обработки события. На первый взгляд здесь кажется уместным использование бинарного дерева либо бинарного поиска. Однако следует учесть, что новое событие в списке не может опуститься в списке ниже, чем на m позиций от вершины, где m – количество конфликтных групп переходов. Если принять длину списка за n, то бинарный поиск позволит вставить элемент на нужную позицию приблизительно за $\log_2 n$, а последовательный обход элементов списка с конца – за m. Выходит, что даже при наличии в сети 10 конфликтующих переходов последовательный обход будет быстрее бинарного при списке больше 1000 элементов.

3) Несмотря на повышение скорости сортировки, даже выборка и добавление новых элементов будет замедлена при большом количестве элементов в управляющем списке. Выход прост – его надо чистить. Нельзя забывать, что надо хранить первое появление каждой маркировки для подсчета времени возврата. Лучше всего вынести их в отдельный массив, а потом, при необходимости, вернуть назад, это ускорит время работы на этапе симуляции.

### Выбор структуры данных для реализации управляющего списка

При реализации управляющего списка на практике, т.е. представлении как структуры данных в каком-либо из языков программирования, нужно иметь возможность сортировать его по определенному полю (чаще всего по времени), выполнять операции произвольного доступа, добавлять и удалять новые элементы по индексу и/или ключу.

Возможные структуры делятся на несколько основных групп – списки (Lists), наборы (Sets), множества (Maps), очереди (Queues) и стек (Stack). Из возможных структур в первую очередь исключено использование наборов, т.к. хоть они и поддерживают автоматическое упорядочивание, в них нельзя содержать объекты с одинаковыми характеристиками, а в управляющем списке гарантировано наличие событий с одинаковым временем выполнения. То же самое касается и множеств, список ключей для доступа к объектам которых тоже является набором. Упорядоченный стек или приоритетная очередь – наилучший по быстродействию вариант, за исключением необходимости сбора статистики после каждого шага, что значительно замедляет выполнение. Остается использование списков.

Рассмотрим два основных типа списка: динамический массив и связный список:

**Таблица 1. Время выполнения стандартных операций**

| | Произвольный доступ | Вставка в конец | Удаление |
|---|---|---|---|
| ArrayList | O(1) | O(1) | O(n) |
| LinkedList | O(n) | O(1) | O(1) |

В таблице использована O-нотация:
- O(1): время выполнения константно.
- O(log n): если n удвоено, то время увеличено на константу.
- O(n): если n удвоено, то время удвоено.

Вставка значительно быстрее в связном списке, а выбор по индексу – в динамическом массиве. Также в связном списке быстрее удаление, что может пригодиться на этапе оптимизации. Поэтому следует рассмотреть оба варианта.

### Реализация способов минимизации управляющего списка и выбор наилучшего

Программа для работы с сетями Петри, включающая режим симуляции работы сетей, написана на языке Java.

UML-диаграмма подсистемы симуляции сетей представлена на рисунке 1.

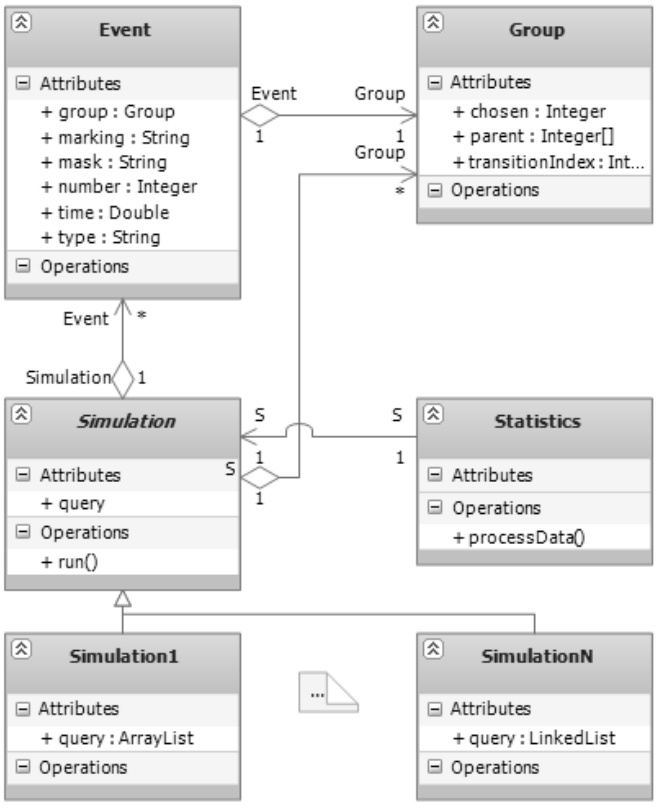


***Рис. 1. Упрощенная UML-диаграмма подсистемы симуляции сетей Петри***

Подсистема собирает статистику по состояниям в процессе моделирования сети для каждой реализации управляющего списка. Рассмотрим 4 способа минимизации списка на основе следующих процедур.

1) Сортировка

Очевидно, что, несмотря на оптимизированный способ сортировки, время работы все равно будет довольно большим, и оно будет экспоненциально расти с увеличением размера списка. Тем не менее, на этом этапе можно сравнить поведение структур ArrayList и LinkedList:

**Таблица 2. Время генерации списка с сортировкой**

| Событий | Время генерации списка, нс | |
|---|---|---|
| | ArrayList | LinkedList |
| 1000 | 89226790 | 95038361 |
| 10000 | 7837538278 | 11300001155 |
| 20000 | 32967832629 | 49836211442 |
| 30000 | 78576589412 | 125722543059 |
| 40000 | 160307454637 | 275728821975 |
| 50000 | 289759595362 | 521567271651 |

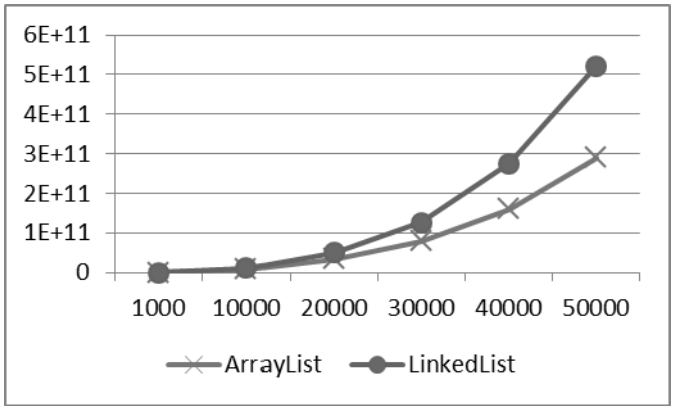


***Рис. 2. Время генерации списка с сортировкой слиянием***

Из графика на рис.2 видно, что структура LinkedList не позволяет минимизировать время генерации управляющего списка. При количестве событий больше 50000 время генерации почти вдвое превышает время при использовании динамического массива. Далее будем рассматривать варианты минимизации, использующие только ArrayList.

2) Прямая вставка на позицию

Вставка всегда будет осуществляться за константное время, полностью зависящее от размера исследуемой сети. Время генерации списка при этом будет строго линейно зависеть от его размера.

**Таблица 3. Время генерации списка с прямой вставкой**

| Событий | Генерация списка, нс | Сбор статистики, нс |
|---|---|---|
| 1000 | 5845663 | 2173635 |
| 10000 | 40214713 | 23834098 |
| 20000 | 108813695 | 45496210 |
| 30000 | 159848742 | 67517939 |
| 40000 | 237567861 | 95970391 |
| 50000 | 279228181 | 115345045 |
| 100000 | 845359073 | 252335708 |
| 1000000 | 8900266852 | 3412606200 |

Мы добились изменения зависимости времени генерации от размера списка с экспоненциальной на линейную. Тем не менее, линейная зависимость означает, что чем больше наш список, тем больше времени будет тратиться впустую.

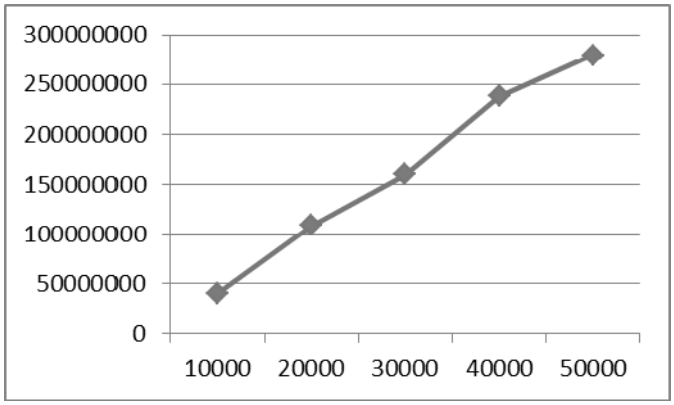


***Рис. 3. Время генерации с прямой вставкой***

Поэтому следующий шаг – непосредственное уменьшение размеров списка.

3) Очищение списка

При расчете оптимального использования данного способа следует исходить из размера списка N, требуемого для очищения списка времени M, коэффициента экспоненты k и искомого количества удаляемых записей x. Аналитически это можно записать в виде неравенства:

$$\frac{N}{x}e^{k*x}+\frac{MN}{x}\leq e^{k*N}$$

Все параметры, кроме N, зависят от языка программирования и способа реализации, поэтому рассчитывать по этой формуле нецелесообразно. К тому же, определение коэффициента k – тоже нетривиальная задача. Поэтому число x лучше искать опытным путем. Минимизация для малого количества событий не имеет смысла, поэтому примем N = 100000 событий, а x от 100 до 2000..

**Таблица 4. Время генерации списка для разного количества очищаемых записей**

| Число x | Генерация списка, нс |
|---|---|
| 100 | 727268279 |
| 200 | 663714751 |
| 500 | 659862872 |
| 1000 | 646911651 |
| 1500 | 657634776 |
| 2000 | 663578382 |

Выберем очищение на каждые 1000 событий. Важно отметить, что нельзя просто очистить список до нуля. Нужно сохранить, во-первых, время первого появления каждой маркировки, а во-вторых, времена запланированных, но не обработанных событий. Анализ статистики происходит при каждом очищении, поэтому требуемое время подсчитать не трудно.

**Таблица 5. Время генерации списка с оптимизацией очищением**

| Событий | Генерация списка, нс | Сбор статистики, нс |
|---|---|---|
| 1000 | 25433949 | 237546 |
| 10000 | 160658817 | 10*237546 |
| 20000 | 340589676 | 20*237546 |
| 30000 | 450885932 | 30*237546 |
| 40000 | 604728491 | 40*237546 |
| 50000 | 776026302 | 50*237546 |
| 100000 | 1501107724 | 100*237546 |
| 1000000 | 15529057904 | 1000*237546 |

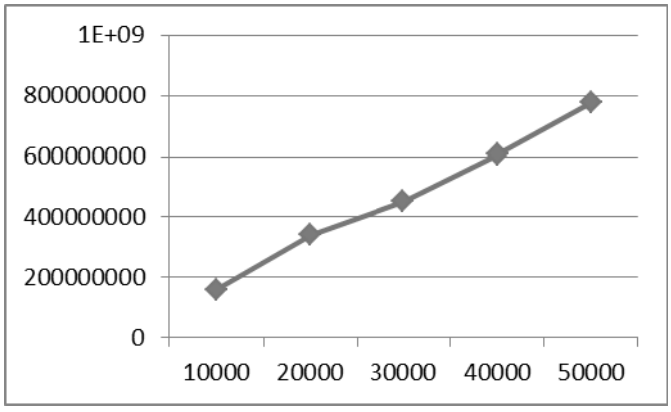


***Рис. 4. Время генерации с очищением списка.***

Зависимость так же выходит линейной, как и в предыдущем методе, но коэффициент наклона кривой больший, т.е. способ менее эффективен.

4) Комбинация методов

Недостаток второго способа можно исправить уменьшением управляющего списка.

Необходимости сортировать даже небольшие участки списка можно заменить вставкой.

Учитывая, что оба способа подходят к решению проблемы с разных сторон, их комбинация должна дать хороший результат.

**Таблица 6. Время генерации списка с комбинированным методом оптимизации**

| Событий | Генерация списка, нс | Сбор статистики, нс |
|---|---|---|
| 1000 | 9650317 | 237546 |
| 10000 | 70032712 | 10*237546 |
| 20000 | 131040315 | 20*237546 |
| 30000 | 201106569 | 30*237546 |
| 40000 | 252545129 | 40*237546 |
| 50000 | 316800955 | 50*237546 |
| 100000 | 654746279 | 100*237546 |
| 1000000 | 6314954473 | 1000*237546 |

## Перспективы

Способы минимизации, рассмотренные в данной статье, позволяют сократить время генерации и обработки управляющего списка. Так же сокращается и общее время моделирования, что позволяет получить результаты статистики по состояниям за меньший период времени, чем без минимизации. Для наглядности, изобразим на графике зависимости времени обработки списков событий от общего числа событий: диаграммы для 2, 3 и 4 способов:


***Рис.5. Три способа минимизации списка***

Было выявлено, что при небольшом количестве событий моделирования системы нет особой необходимости в минимизации, но для моделирования реальных систем необходимо количество событий 10 000 и более, что требует больших затрат времени.

Для проведения данных исследований была разработана среда эмуляции стохастических

сетей Петри на языке Java. В нее были внесены рассмотренные способы оптимизации и модуль проверки быстродействия. Программа размещена в свободном доступе на сайте https://sourceforge.net/projects/petrineteditor/, что дает право любому желающему изучить, модифицировать или дополнить исходные коды, а так же предложить свои варианты оптимизации на форуме этой программы.

В дальнейшем на базе этой программы будет разработан модуль, позволяющий не просто моделировать различные сети Петри, но делать это максимально эффективно, с анализом времени выполнения и возможностью заменять любую часть алгоритма более эффективной.

Учитывая, что программа написана на кросс платформенном языке, ее можно будет использовать на любых компьютерах. Поэтому следующий шаг после минимизации списка событий – настройка Petri-nets системы имитационного моделирования для работы в распределенных компьютерных системах.

## Список литературы

1. M. Ajmone Marsan, G. Balbo, G. Conte, S. Donatelli and G. Franceschinis «Modelling with Generalized Stochastic Petri Nets» John Wiley and Sons pp.117-155
2. Molloy M. K. "Performance Analysis Using Stochastic Petri Nets", IEEE Trans. on Computers, vol. C-31. No. 9, pp. 913-917
3. Peterson J.L. Petri Net Theory and the Modeling of Systems, Prentice-Hall, 1981
4. Kumar, D. and Harous, S., "Distributed Simulation of Timed Petri Nets: Basic Problems and Their Resolution", IEEE Transactions on Systems, Man and Cybernetics, Vol. 24, No. 10
5. Holliday, M. A. and Vernon, M. K., "A Generalized Timed Petri Net Model for Performance Analysis", IEEE Trans. on Software Eng., vol. SE-13, No. 12, pp. 1297-13