Санкт-Петербургский государственный университет информационных технологий, механики и оптики

Кафедра «Компьютерные технологии»

М. А. Коротков

АЛГОРИТМЫ УКЛАДКИ ДИАГРАММ СОСТОЯНИЙ

Магистерская диссертация

Руководитель: канд. физ.-мат. наук Ф.А. Новиков

Санкт-Петербург
2007

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

Программный пакет с открытым исходным кодом *UniMod* [1] обеспечивает разработку и выполнение автоматно-ориентированных программ. Разработанный пакет базируется на парадигме автоматного программирования [2]. Он позволяет создавать и редактировать диаграммы классов и состояний *UML* [3], которые соответствуют графу переходов и схеме связей конечного автомата [4]. После описания конечного автомата с помощью диаграмм существует возможность запустить его. При этом содержимое диаграмм преобразуется в *XML*-описание, которое передается интерпретатору, также входящему в пакет *UniMod*.

Во многих случаях возникает потребность автоматической укладки диаграмм. В частности, в первых версиях *UniMod* отсутствовал графический редактор и, соответственно, в файлах не содержалось информации о расположении элементов, поэтому, при загрузке сохраненных файлов предыдущих версий, необходимо обеспечить автоматическую укладку. В рамках проекта *UniMod* основной интерес представляет укладка диаграмм состояний языка *UML*.

При укладке диаграммы необходимо учитывать ряд критериев, которые могут противоречить друг другу, например, такие как минимизация площади, занимаемой диаграммой, и наличие достаточного количества свободного места («воздуха»). Каждый критерий оценивает «качество» диаграммы для того или иного применения (отображения на мониторе, печати и т.д.). Такие критерии называются **эстетиками**. Задавшись некоторым набором эстетик, можно построить штрафную функцию (которая тем больше, чем менее построенная укладка соответствует выбранным критериям), и пытаться минимизировать её, перемещая элементы, или разработать алгоритм, последовательно модифицирующий исходную диаграмму так, чтобы результат соответствовал выбранным критериям.

1. ПОСТАНОВКА ЗАДАЧИ

Рассмотрим диаграмму состояний конечного автомата. Она включает в себя состояния и переходы (и у тех и у других есть дополнительные, связанные с ними, атрибуты – метки, но сейчас не стоит останавливаться на этом вопросе). Диаграмме состояний (без вложенных состояний) можно сопоставить граф [5] (при укладке диаграммы состояний не важна ориентация дуг). Каждому состоянию соответствует вершина, а переходу – ребро. При этом необходимо несколько сузить поле деятельности, зафиксировав представление элементов и тип носителя, на котором будет изображаться диаграмма.

1.1. Выбор типа носителя и представления элементов

Диаграмма может быть изображена на плоскости или может быть построена объемная модель [6]. Для представления на мониторе персонального компьютера наиболее естественным является выбор плоского носителя.

Выбор вида элементов графа называют **изобразительным соглашением** [7].

В работе [7] приведен ряд наиболее популярных изобразительных соглашений. Перечислим некоторые из них:

- **полилинейное изображение** – каждое ребро изображается в виде ломаной линии;
- **прямолинейное изображение** – каждое ребро представляется с помощью отрезка прямой;
- **ортогональное изображение** – каждое ребро графа изображается в виде ломаной линии, состоящей из вертикальных и горизонтальных отрезков;
- **сетчатое изображение** – все вершины, точки пересечения и изгибы ребер имеют целочисленные координаты.

Кроме того, в изобразительном соглашении описывается представление вершины (окружность, прямоугольник, точка). В дальнейшем будем изображать вершину как прямоугольник, а ребро – как ломаную линию с конечным числом изломов (полилинейное изображение).

Выбор такого вида элементов объясняется эстетическими соображениями, а также традициями, сложившимися в области построения диаграмм.

Описанный выбор вида элементов диктует использование декартовой системы координат. Вершину принято изображать как прямоугольник со сторонами, ориентированными параллельно координатным осям. Для изображения ее достаточно знать координаты левого верхнего угла, ширину и высоту, а для изображения ребра – координаты его начала, конца и всех точек излома.

1.2. Понятие укладки

Укладкой графа *G = (V; E)* в декартовой системе координат *(X; Y)* назовем множество $L = (G;\ F_V;\ F_E)$, где F_V – функция из множества вершин в множество параметров, необходимых для представления вершины в выбранной системе координат (в нашем случае $F_V : V \to X \times Y \times R \times R$, где последняя пара параметров – ширина и высота прямоугольника). F_E – функция из множества ребер в множество параметров, необходимых для представления ребра в выбранной системе координат (в нашем случае $F_E : E \to (X \times Y)^n, n \in N$, где параметр *n* – количество изломов, вообще говоря может быть также переменным). Для простоты будем говорить, что в уложенном графе заданы геометрические параметры каждого элемента.

Задача построения визуального изображения диаграммы по известным параметрам ее элементов в заданной системе координат (по заданной укладке) – аспект реализации, не затрагиваемый в данной работе.

Две укладки *L, L'* назовем **изоморфными**, если одна получается из другой путем соответствующего изменения начала координат. Далее будем называть изоморфные укладки одинаковыми. Укладки на рис. 1, например, являются различными.

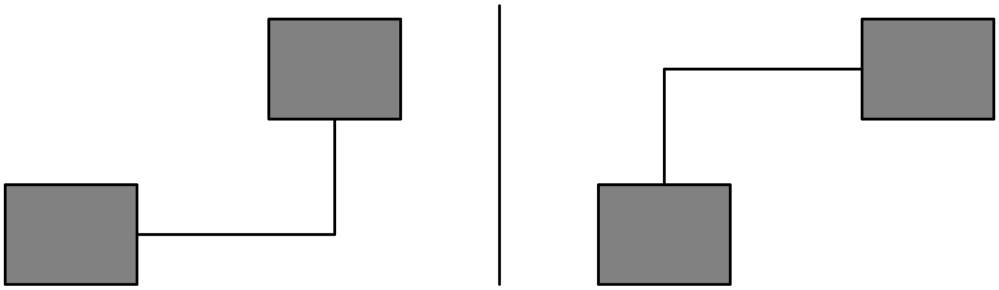

Рис. 1. Различные укладки

Алгоритм укладки диаграммы должен строить пару функций укладки $\{F_V;\ F_E\}$, или сопоставлять соответствующие наборы геометрических параметров (координаты, линейные размеры, точки изломов) каждой вершине и ребру.

Укладку на плоскости будем называть **плоскостной.** Данный термин не является общепринятым, в литературе (например, в работе [8]) чаще говорят просто об изображении графа на плоскости. Укладку, в которой ребра представляют собой ломаные, состоящие только из горизонтальных и вертикальных отрезков, будем называть **плоскостной ортогональной** (или просто **ортогональной**). Ортогональная укладка соответствует соглашению об ортогональном изображении.

1.2.1. Общий подход к оценке

Теперь необходимо оценить качество укладки (точнее качество изображения, полученного по некоторой укладке). Выделим набор эстетик, по которым может оцениваться качество укладки. Основная задача такого выделения – обеспечить «читаемость» графа (однозначность представления информации). В работе [9] были выделены следующие эстетики:

- **минимизация числа пересечений**: число пересечений ребер, прохождений ребер по вершинам и наложений вершин должно быть минимальным;
- **минимизация площади**: площадь, занимаемая выпуклой оболочкой уложенного графа (более простой, хотя и не эквивалентный исходному, вариант данной эстетики – площадь минимального прямоугольника, включающего в себя уложенный граф), должна быть минимальной;
- **ограничение «свободного места»**: доля свободного места на диаграмме не должна быть меньше некоторого предела;
- **минимизация изломов**: число изломов должно быть минимально;
- **минимизация общей длины ребер**: суммарная длина ребер должна быть минимальна;
- **минимизация коэффициента сторон**: отношение длины большей стороны к длине меньшей стороны объемлющего граф прямоугольника должно быть минимально;
- **унификация длин ребер**: минимизация различий между длинами ребер на графе.

Пытаясь выразить выбранные эстетики более точно, мы придем к построению штрафной функции (разд. 1.3.1), причем некоторые эстетики распадутся на несколько составляющих этой функции с различными весами – например, штраф за наложение вершин будет значительно больше, чем штраф за пересечение ребер.

В результате анализа результатов работы алгоритмов, созданных в рамках работы [9], а также новых алгоритмов, следующие эстетики были отброшены как менее значимые:

- **минимизация коэффициента сторон;**
- **унификация длин ребер.**

Кроме того, следующие критерии в большой степени учитываются в других критериях и нарушаются только в очень специфических алгоритмах ([10]):

- **минимизация изломов;**
- **минимизация общей длины ребер.**

К примеру, укладка слева на рис. 2 значительно лучше читается, чем укладка справа, поскольку укладка справа не удовлетворяет эстетикам минимизации числа пересечений, количества изломов и унификации длин ребер.

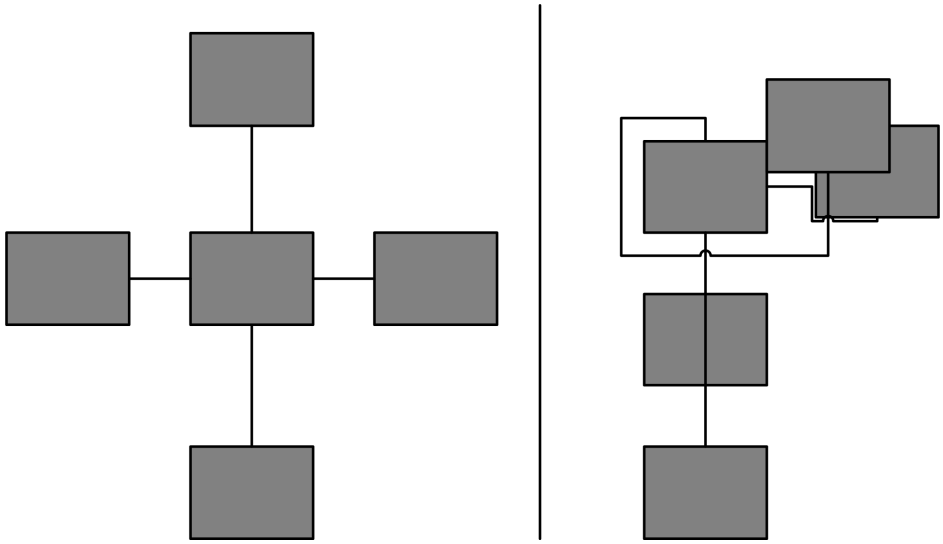

Рис. 2. Укладки графа

Затем, был выделен новый критерий (который, в неявном виде, предполагался и ранее): **выравнивание вершин.**

Укладка слева на рис. 3 отличается от укладки справа как раз тем, что на ней произведено выравнивание вершин друг относительно друга. Выравнивание может производиться как между вершинами, так и по ячейкам достаточно крупной (размер ячейки сопоставим с размером вершины) координатной сетки.

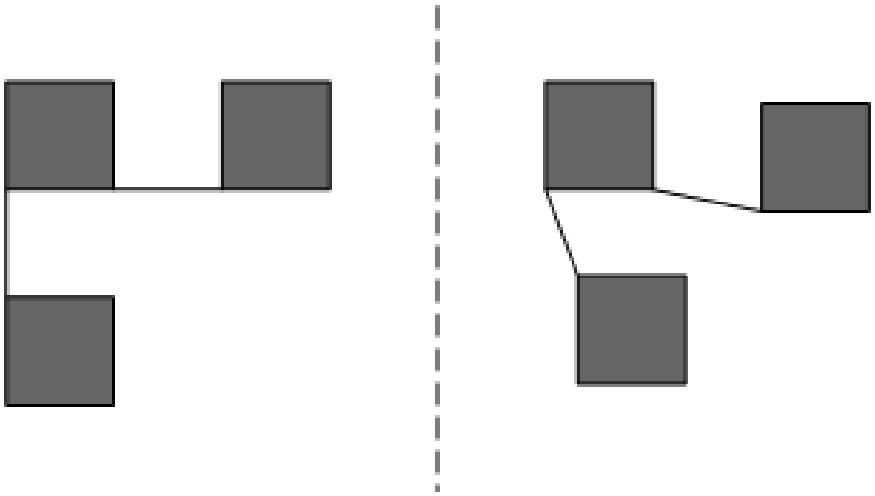

Рис. 3. Выравнивание

Граф, соответствующий диаграмме состояний, отличается от обычного графа возможностью вложения состояний друг в друга, а также наличием выделенных начальных и конечных состояний. Для оценки качества его укладки необходимо учесть дополнительные критерии:

- **унификация размеров вершин, соответствующих простым состояниям**: размер вершин для всех простых состояний (состояний, не имеющих вложенных состояний) необходимо максимально приблизить к заданному оптимальному размеру;
- **разнесение начального и конечного состояний**: расстояние между начальным и конечным состоянием должно быть достаточно велико.

Приведенные выше критерии практически покрывают набор критериев из работы [7] (те из них, которые применимы к ортогональной укладке). Неохваченными остались критерии построения по возможности симметричного изображения и унификации количества изломов на ребре. Первый критерий сложен для оценивания и учета, учет второго не приводит к значительному улучшению читаемости.

Сравним две укладки на рис. 4. Укладка справа более предпочтительна, поскольку на ней переходы «визуально» выполняются слева направо (от начального состояния к конечному). Заметим, что приведенные эстетики, в отличие от штрафной функции, не позволяют количественно оценить укладу и лишь дают понять, какими соображениями будут использоваться при оценке результатов работы программ укладки.

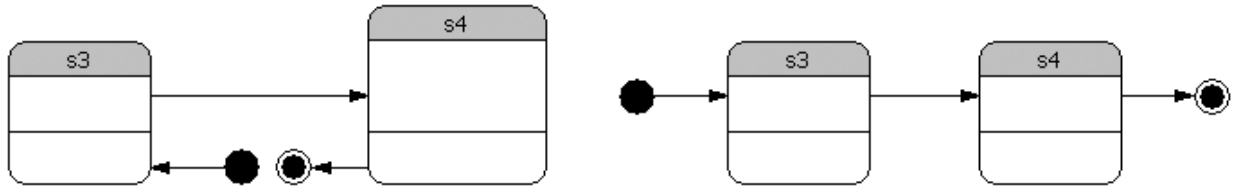

Рис. 4. Укладки диаграммы состояний

Обратим внимание на то, что применяемые в диаграмме состояний *UML* начальное и конечное состояния, не являются прямоугольными. Для простоты заменим их объемлющими прямоугольниками.

Задача, которая ставится в настоящей работе, состоит в построении качественной ортогональной укладки диаграммы состояний. Наличие алгоритма укладки принципиально важно не только для редактора диаграмм, но и для организации импорта файлов, созданных в других редакторах, так как для *UML* редакторов не существует унифицированного механизма обмена данными, сохраняющего информацию о местоположении элементов [11].

Окончательный список критериев, наиболее значимых при рассмотрении алгоритмов данной работы:

- **минимизация числа пересечений**;

  - минимизация пересечений ребер;
  - минимизация прохождения ребер по вершинам;
  - минимизация наложения вершин;
- **минимизация площади**;
- **ограничение «свободного места»**;
- **выравнивание вершин;**
- **унификация размеров вершин, соответствующих простым состояниям**.

1.3. Типы алгоритмов укладки графов

Задача укладки диаграммы не имеет и не может иметь универсального решения, в связи с тем, что набор критериев, применяемых для оценки качества укладки, зависит от типа диаграммы. В работе [10] приведена классификация алгоритмов, применяемых для решения данной задачи. Рассмотрим две группы алгоритмов укладки графов, принципиально отличающиеся подходом к решению:

- алгоритмы с физическим аналогом [12];
- аналитические алгоритмы [8].

1.3.1. Алгоритмы с физическим аналогом

Алгоритмы этой группы ставят в соответствие графу некоторую физическую модель, например, систему пружин, которые стремятся сжаться до некоторой заданной длины (такие алгоритмы называют «пружинным методом») или систему стержней и шарниров, с вершинами – одноименными электрическими зарядами.

Для описания физической модели вводится понятие штрафной функции, задающей потенциальную энергию системы (такие алгоритмы еще называют алгоритмами минимизации потенциала). При этом задача укладки преобразуется в задачу нахождения минимума этой функции, которая решается с помощью сдвига на некоторый вектор каждой вершины графа и проверки изменения значения штрафной функции. Сдвиг вершин выполняется в цикле, условием выхода из которого является либо достижение локального минимума, либо достижение максимума числа допустимых итераций.

Наиболее популярная подгруппа группы алгоритмов с физическим аналогом – **методы отжига**. Они выделяются тем, что «колебания» системы затухают с каждой итерацией. Название этой группы алгоритмов объясняется именно этой особенностью – затухания колебаний эквиваленты постепенному снижению температуры системы. В главе 4 метод отжига рассматривается более подробно.

Для минимизации штрафной функции также можно использовать генетические алгоритмы. В работе [13] приводится более подробное описание такого подхода. Генетические алгоритмы в данном случае используются именно как один из методов минимизации штрафной функции и не являются качественно иным подходом.

1.3.2. Аналитические алгоритмы

Аналитические алгоритмы, в отличие от алгоритмов с физическим аналогом представляют собой последовательность различных преобразований графа, приводящую к построению укладки. Это позволяет получать с их помощью гарантированный результат, удовлетворяющий выбранным критериям, так как критерии используются не для построения штрафной функции, а для построения самого алгоритма укладки, заметим, что на некоторых этапах аналитического алгоритма могут использоваться и псевдослучайные методы, в том числе методы минимизации функций, аналогичные методу отжига.

К недостаткам аналитических алгоритмов можно отнести сложность их построения. Кроме того, они плохо поддаются модификации, и для постановки экспериментов приходится вносить серьезные изменения в сам алгоритм, а не в набор параметров, как это можно делать в случае использования, например, алгоритмов, базирующихся на штрафных функциях.

На практике (рассмотрено три реализации: [14 – 16]) аналитические алгоритмы позволяют относительно быстро получать наиболее качественные и красивые графы. Поэтому для укладки диаграммы состояний в проекте *UniMod* было принято решение использовать эту группу алгоритмов в качестве основной.

Наиболее эффективными [17] в группе аналитических алгоритмов являются алгоритмы группы *GIOTTO* [8, 18] (собственно алгоритм *GIOTTO* и его модификации). В работе [19] приводится обоснование применения модификации алгоритма *GIOTTO* для решения задачи укладки диаграммы состояний. Основная часть работы [19] посвящена укладке реберных меток, однако данная проблема здесь затрагиваться не будет. В работе [8] изложены основы алгоритма *GIOTTO*. Прежде чем перейти к собственным алгоритмам укладки, рассмотрим некоторые из существующих решений.

2. ОБЗОР ИЗВЕСТНЫХ РЕШЕНИЙ

Задача укладки диаграммы состояний решается многими *CASE* средствами, поддерживающими язык *UML*. Рассмотрим наиболее популярные из них. Все примеры, приведенные в текущем разделе, построены для одной и той же диаграммы состояний.

Далее описаны особенности укладки в следующих продуктах:

- *Rational Rose Professional J Edition;*
- *Borland Together Designer CE;*
- *NetBeans 5.5;*
- *yFiles;*
- *AGD.*

2.1. Укладка диаграмм в *Rational Rose Professional J Edition*

В рассматриваемом продукте использован один алгоритм укладки (либо осуществляется автоматический выбор алгоритма). На рис. 5 приведена уложенная с его помощью диаграмма состояний. Эстетическое качество укладки неудовлетворительно: неоправданные пересечения переходов, прохождения переходов по состояниям, при наличии петель не удается обеспечить читаемость диаграммы. Возможно, использованный алгоритм более подходит для укладки диаграммы классов (подробнее об этом в [20]).

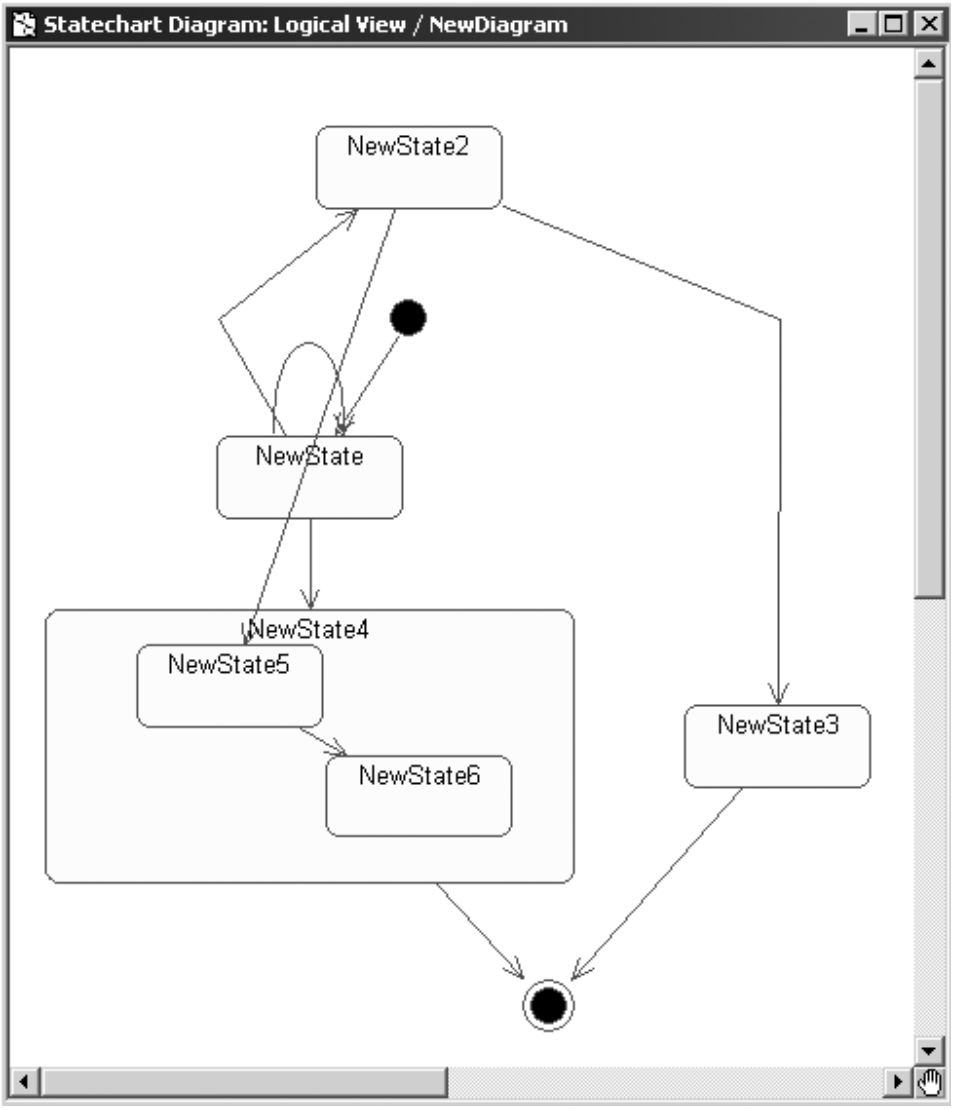

Рис. 5. Укладка в *Rational Rose*

2.2. Укладка диаграмм в *Borland Together Designer CE*

Данный продукт предоставляет ряд алгоритмов укладки диаграмм:

- иерархическая укладка;
- укладка деревьев;
- метод отжига;
- ортогональная укладка;

- оригинальный алгоритм *Together*.

Алгоритмы иерархической укладки и укладки деревьев могут быть отброшены без рассмотрения, как неподходящие для применения к диаграмме состояний. Рассмотрим оставшиеся алгоритмы.

Метод отжига, реализованный в рассматриваемом программном продукте, не представляет особого интереса (достаточно посмотреть на рис. 6). Укладка занимает неоправданно много места.

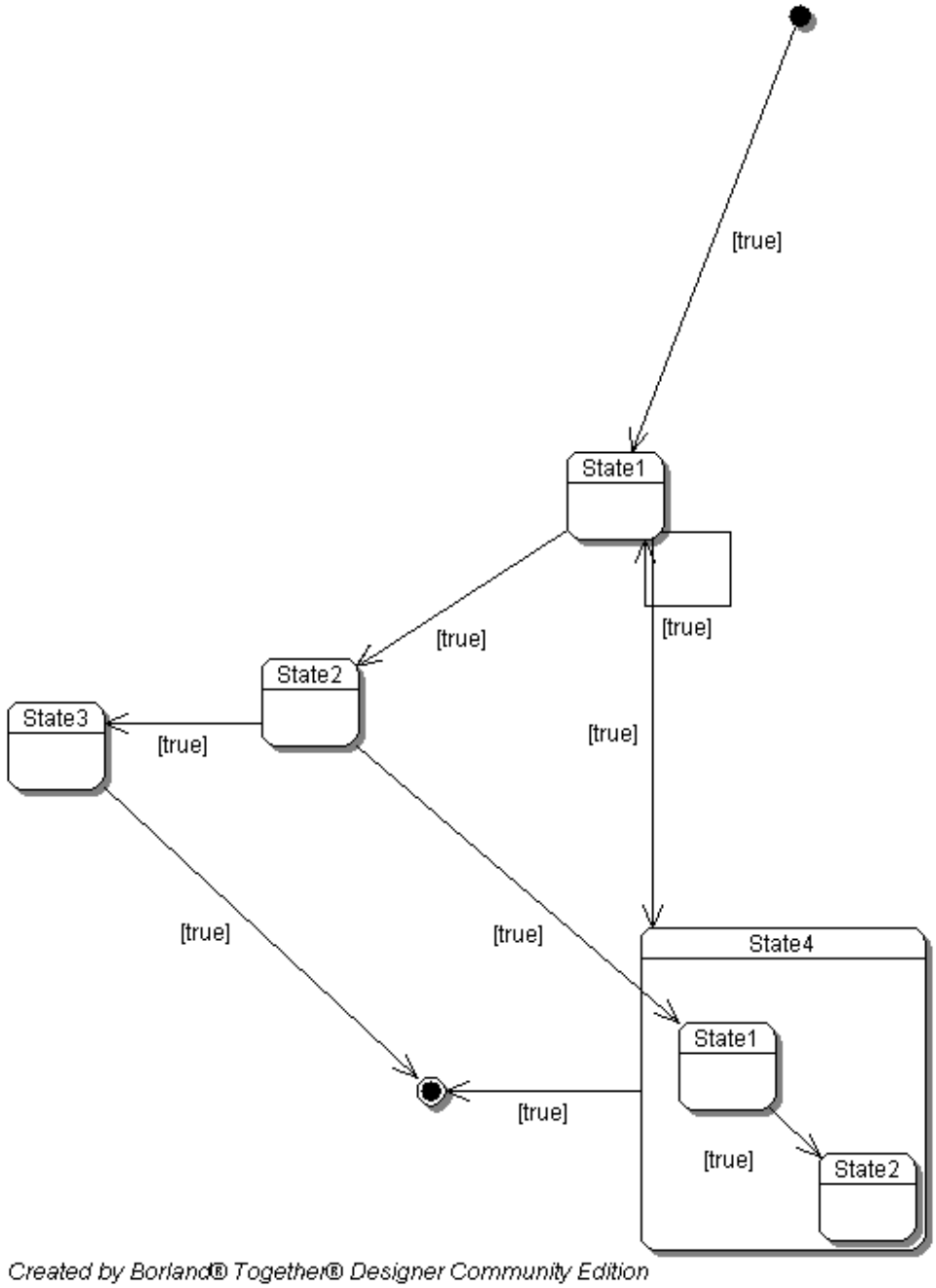

Рис. 6. Метод отжига

Алгоритм ортогональной укладки, вероятно, является модификацией простого алгоритма ортогонализации, притом достаточно неудачной. Переходы во вложенные состояния не удовлетворяют требования ортогональной укладки. Количество пересечений неоправданно велико (рис. 7).

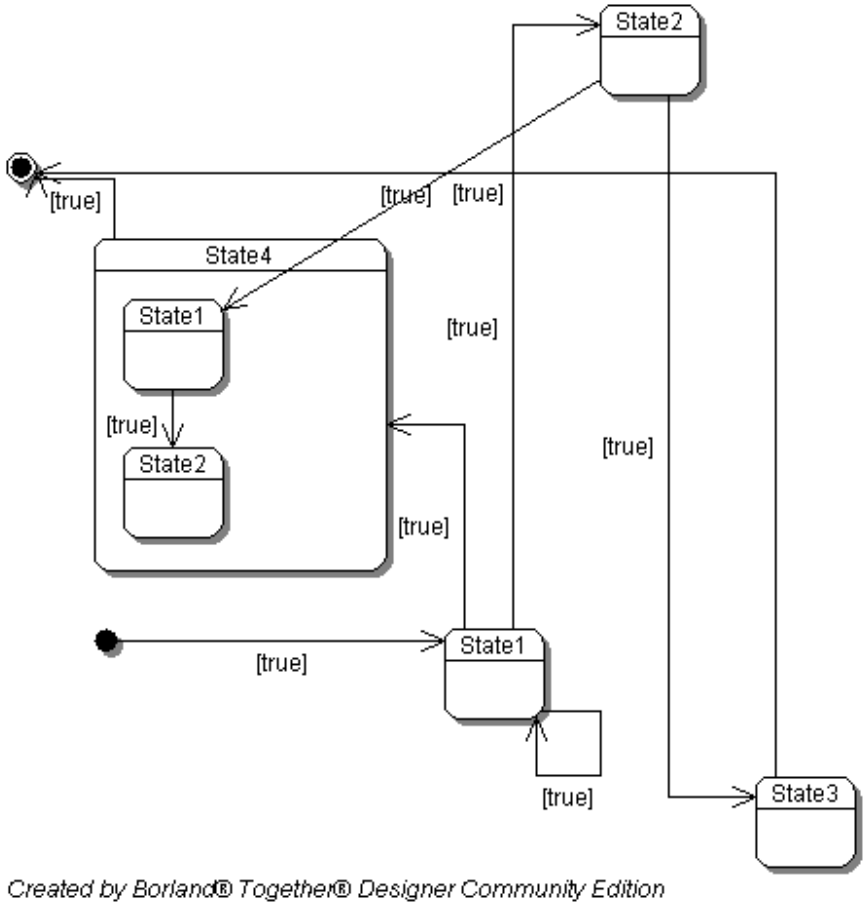

Рис. 7. Ортогональная укладка

На рис. 8 приведен результат укладки с использованием оригинального алгоритма *Together*. Последний осуществляет неортогональную укладку, достигая высокой компактности диаграммы. Эстетически укладки, построенные им, кажутся достаточно качественными. Информация о том, на базе каких идей построен алгоритм, компанией *Borland* не разглашается.

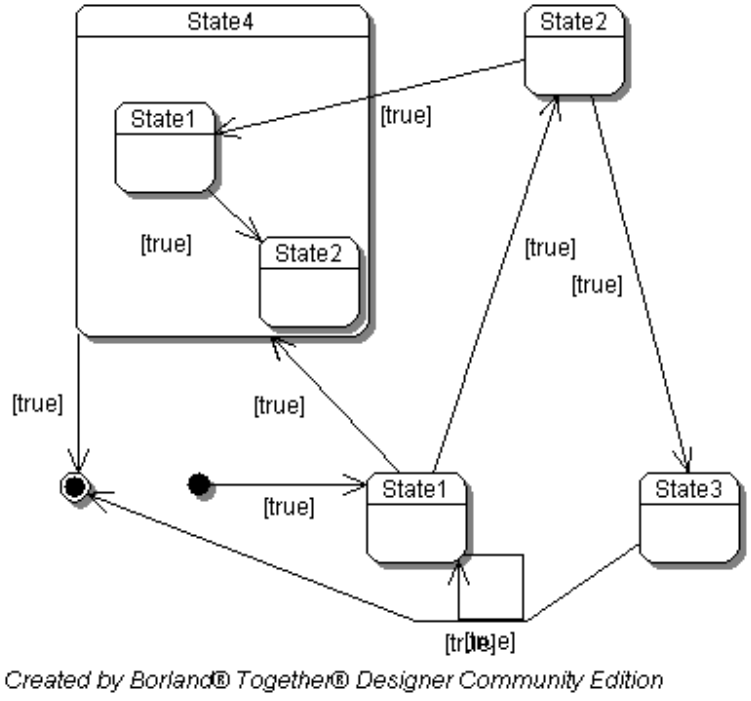

Рис. 8. Алгоритм *Together*

2.3. Укладка в *NetBeans 5.5*

Продукт *NetBeans 5.5*, является открытым, но разрабатывается и поддерживается компанией Sun Microsystems. Алгоритмы укладки, предоставляемые модулем *UML Modeling*, перешли практически без изменений из *Sun Java Studio Enterprise 7*.

В продукте *NetBeans 5.5 (*с установленным модулем *UML Modeling)* доступно четыре вида укладки:

- «иерархическая» укладка;
- ортогональная укладка;
- «симметричная» укладка;
- инкрементальная укладка.

«Иерархическая» укладка (**рис. 9**) дает недостаточную компактность укладки и не является ортогональной. Ортогональная укладка (**рис. 10**), тоже не является в действительности ортогональной (возможно, не решена проблема укладки вложенных состояний). «Симметричная» укладка (**рис. 11**) достаточно компактна, но допускает прохождение ребер по состояниям. Инкрементальная укладка (**рис. 12**) вероятно реализует один из алгоритмов отжига, причем не очень глубоко (после первого запуска, с каждым последующим результат заметно меняется). Для проверки было проведено порядка 50 запусков укладки подряд. Результат: есть немотивированное пересечение. Кроме того, на составном состоянии закладка с именем состояния находится «снаружи», что противоречит рекомендациям стандарта и не произведено выравнивание вершин.

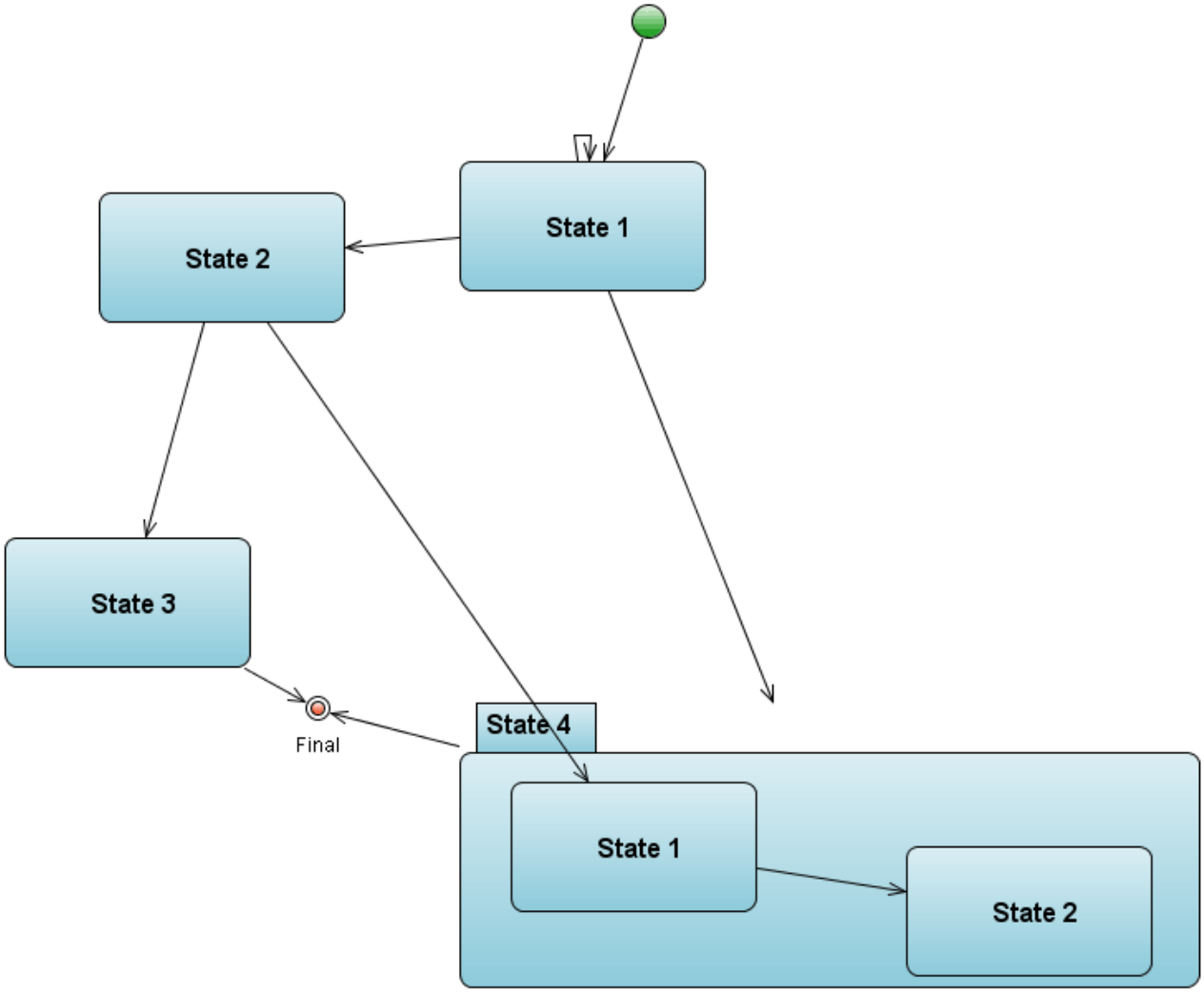

Рис. 9. «Иерархическая» укладка в *NetBeans 5.5*

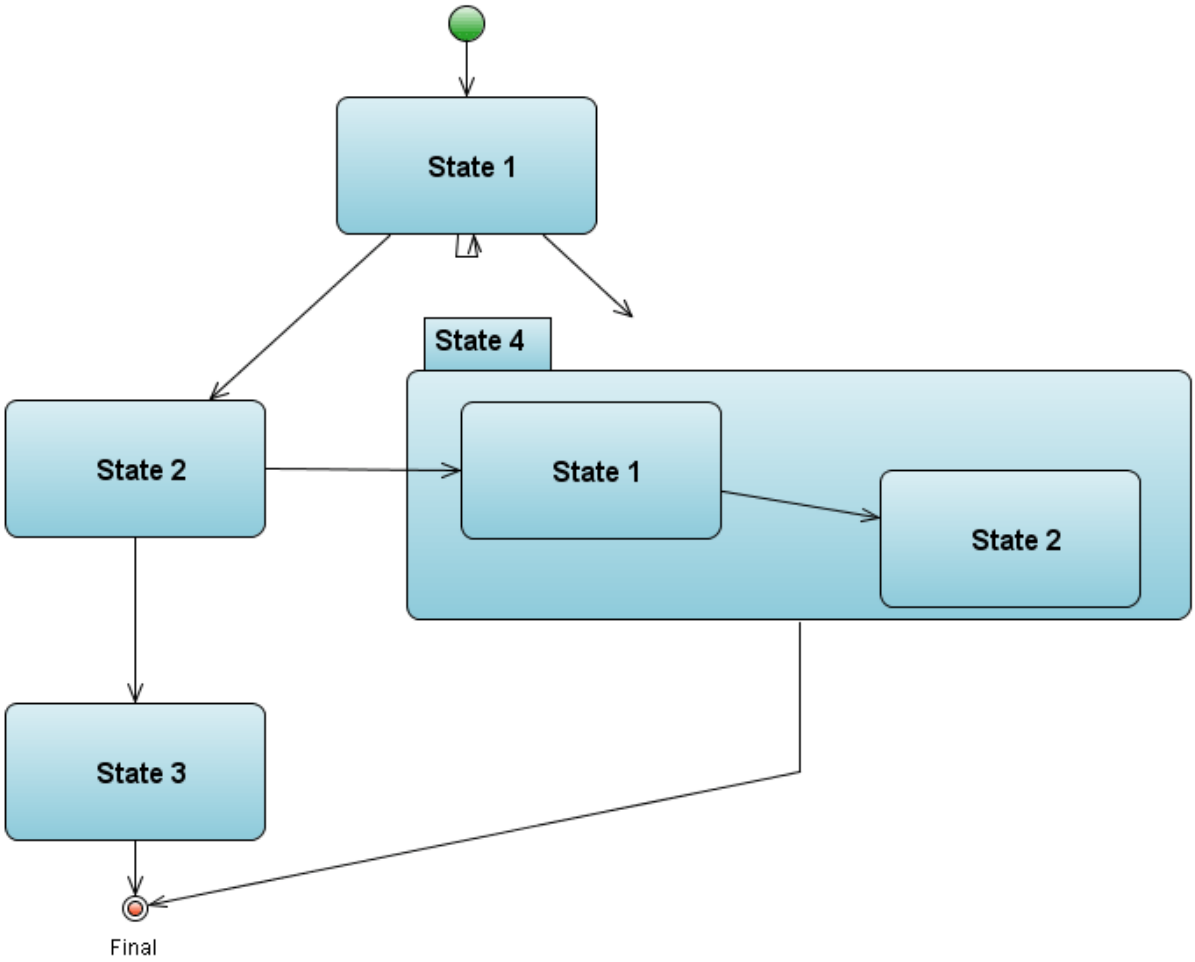

Рис. 10. Ортогональная укладка в *NetBeans 5.5*

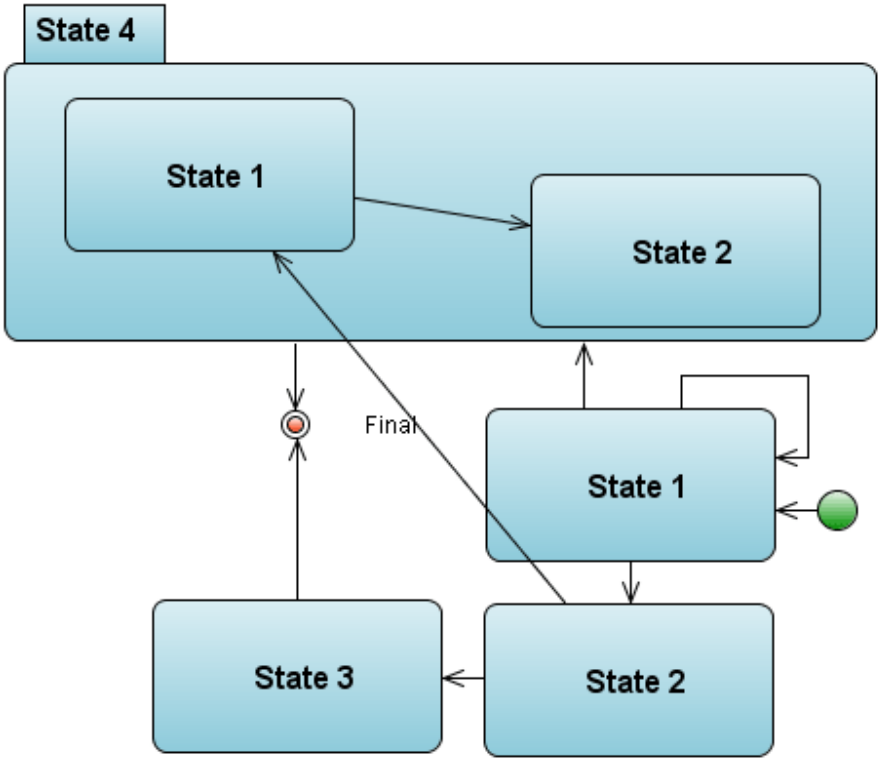

Рис. 11. «Симметричная» укладка в *NetBeans 5.5*

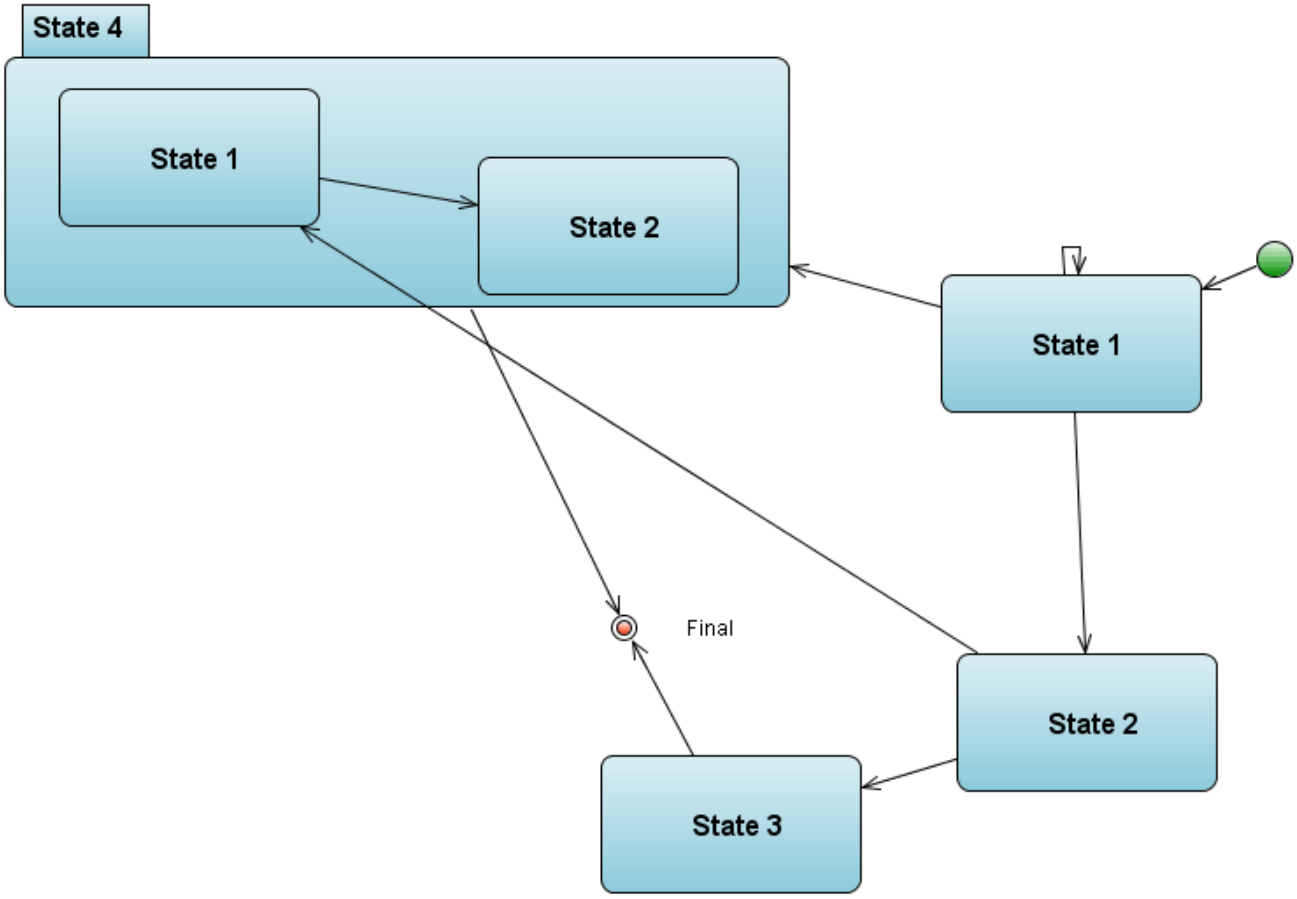

Рис. 12. Инкрементальная укладка в *NetBeans 5.5*

2.4. Укладка с помощью пакета *yFiles*

Комплекс алгоритмов укладки графов *yFiles* применен, например, в *UML*-редакторе *Poseidon for UML*. К сожалению, в некоммерческой версии этого продукта укладка диаграмм недоступна. Поэтому на рис. 13 приведена укладка графа, соответствующего диаграмме состояний использованной в остальных примерах, выполненная с помощью редактора графов *yEd*. Компания-разработчик не разглашает идей, на которых основан алгоритм. Обратим внимание на то, что граф уложен без пересечений и укладка является ортогональной.

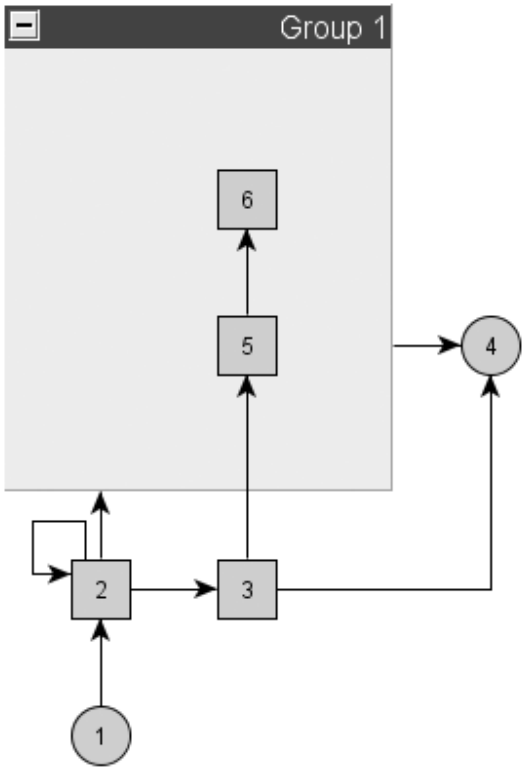

Рис. 13. Укладка с помощью *yFiles*

2.5. Укладка графа с помощью пакета *AGD*

Для укладки графов (без группировки вершин) разработан ряд пакетов, демонстрирующих возможности современных алгоритмов. На рис. 14 приведен результат укладки графа в программе *AGD* [14]. Укладка производилась с использованием алгоритма *GIOTTO*. Поддерживаются только планарные графы.

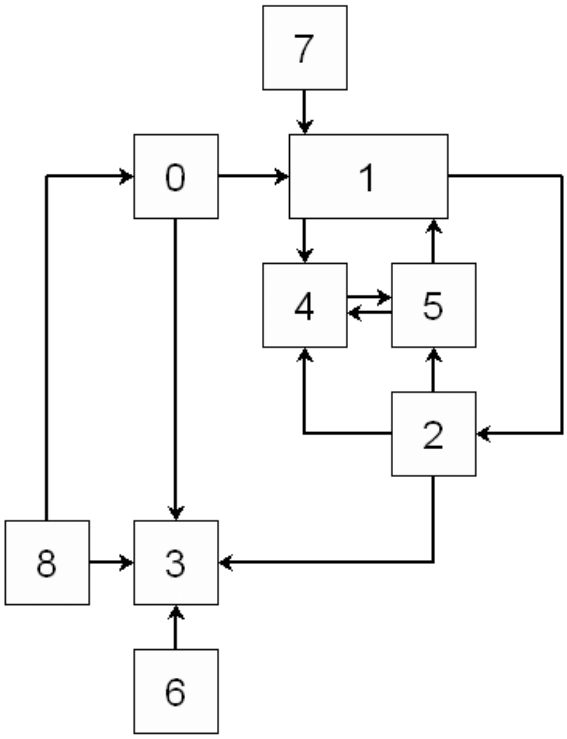

Рис. 14. Укладка с помощью *AGD*

2.6. Результаты обзора

Проведенный обзор показывает, что наиболее популярные на данный момент средства редактирования *UML* диаграмм не используют качественных алгоритмов укладки диаграмм состояний (и, если такие алгоритмы существуют, то они сами недостаточно популярны). С другой стороны, для укладки графов общего вида разработан целый ряд алгоритмов, которые строят хорошие укладки. Поэтому работа по анализу применимости и адаптации существующих алгоритмов укладки графов к задаче укладки диаграмм состояний является актуальной и практически значимой.

3. СУЩЕСТВУЮЩИЕ АЛГОРИТМЫ

3.1. Вспомогательные алгоритмы

В дальнейшем, нам понадобятся следующие вспомогательные алгоритмы:

1. Алгоритм построения связного надграфа.
2. Алгоритм построения двусвязного надграфа.
3. Алгоритм построения *st*-графа.

Алгоритмы 1 и 2 производят добавление фиктивных ребер в граф. В случае если алгоритм укладки работает только со связными (двусвязными) графами, перед применением алгоритма укладки используется соответствующий алгоритм с добавлением ребер, а на этапе визуализации фиктивные ребра не визуализируются. В дальнейшем на этих действиях внимание не акцентируется.

3.1.1. Построение связного графа

Для того чтобы сделать граф связным, найдем его компоненты связности [21], упорядочим их произвольным образом (получим список компонент) и соединим произвольные вершины «соседних» компонент в списке мостами (пример приведен на рис. 16). В исходном графе на рис. 15 три компоненты связности. Добавленные в ходе работы алгоритма ребра выделены на рис. 16 пунктиром. Для графа с N компонентами связности будет добавлено $N-1$ фиктивное ребро.

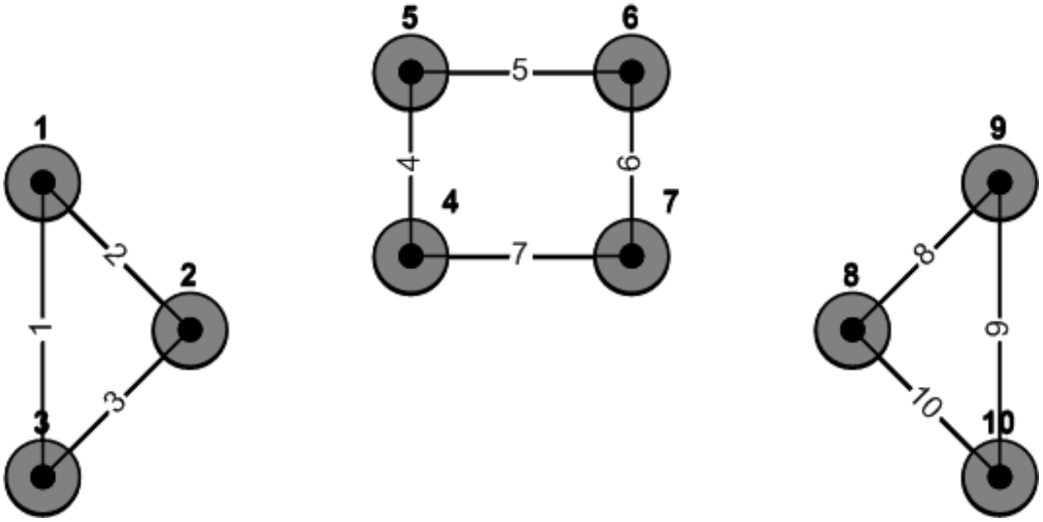

Рис. 15. Исходный граф

В рамках рассматриваемой задачи, можно также учесть, что граф переходов конечного автомата не может являться несвязным (с другой стороны, реализация этой части алгоритма необходима для обобщения его на другие задачи).

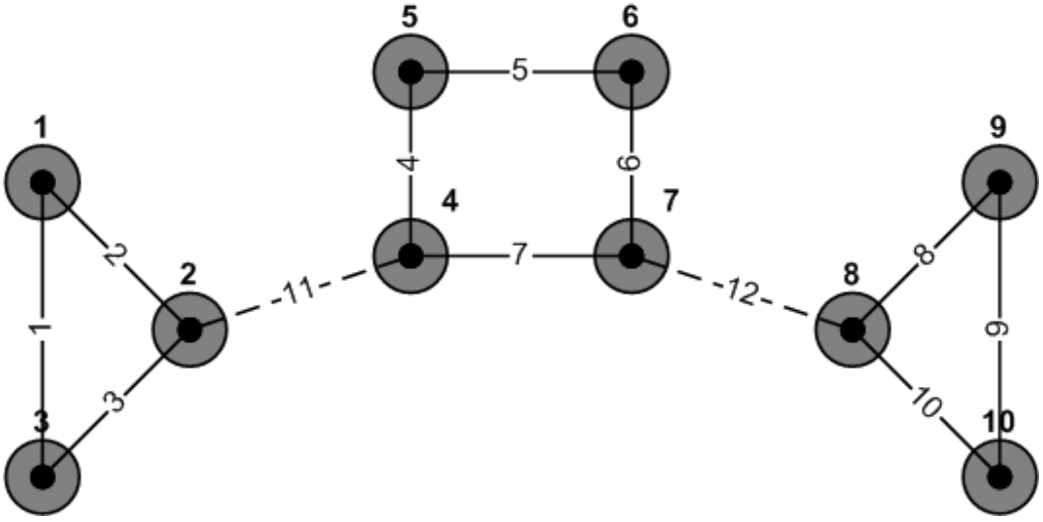

Рис. 16. Построение связного графа

3.1.2. Построение двусвязного надграфа

Двусвязный граф (**дираф**) остается связным после удаления любого ребра. В работе [22] описан ряд алгоритмов построения двусвязных графов. Воспользуемся следующим алгоритмом: выделим компоненты двусвязности, построим дерево таких компонент и прошьем его крону циклом (рассматривая двусвязные компоненты как отдельные вершины). Приведем более строгое описание этого алгоритма:

- возьмем связный граф G;
- выделим в нем двусвязные компоненты (**блоки**), точки сочленения и мосты. В графе на рис. 16 ребра e_{11}, e_{12} – мосты. Граф содержит три блока: $x_1 = \{v_1, v_2, v_3\}$, $x_2 = \{v_4, v_5, v_6, v_7\}$, $x_3 = \{v_8, v_9, v_{10}\}$;
- построим **граф блоков** $B(G)$. Вершины этого графа соответствуют блокам, мостам и точкам сочленения исходного графа. В графе блоков существует ребро $e = (v, v')$, если:
  - v соответствует точке v_c сочленения в графе G, v' – блоку x, содержащему данную точку сочленения;
  - v соответствует мосту e_m в графе G, v' – блоку, содержащему одну из вершин этого моста.

Граф блоков для рассматриваемого примера приведен на рис. 17.

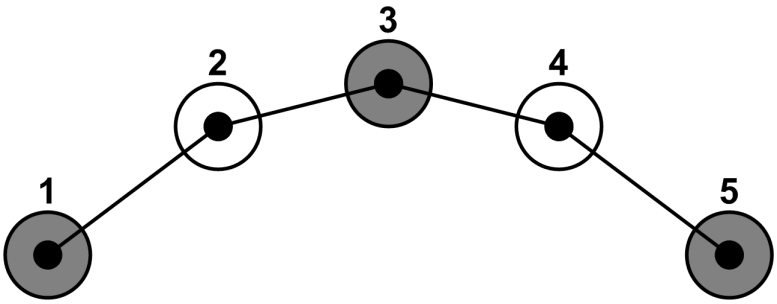

Рис. 17. Граф блоков

Вершины v_1, v_3, v_5 соответствуют блокам x_1, x_2, x_3, а v_2, v_4 – мостам e_{11}, e_{12}. Заметим, что граф блоков всегда является свободным деревом [23]. Выделим все висячие вершины и соединим их в цепочку (как и в алгоритме построения связного графа, для N вершин понадобится $N-1$ ребро). В рассматриваемом примере будет добавлено одно ребро (рис. 18). На этом рисунке фиктивное ребро выделено пунктиром.

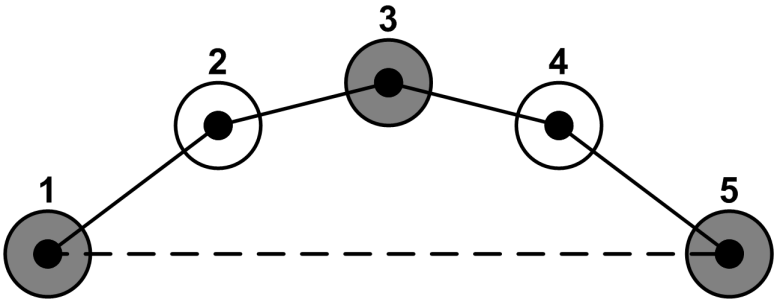

Рис. 18. Граф блоков после преобразования

Вершинам степени 1 в графе блоков всегда соответствуют блоки в исходном графе, причем в каждом таком блоке найдется хотя бы одна вершина, не являющаяся точкой сочленения. Обозначим такую вершину $B^{-1}(v)$, где v – вершина в графе блоков. Для каждого фиктивного ребра $e = (v_1, v_2)$, добавленного в граф блоков, добавим ребро $e' = (B^{-1}(v_1), B^{-1}(v_2))$ в исходный граф. В результате получим граф, изображенный на рис. 19. Пунктиром выделено ребро, добавленное на этапе построения двусвязного графа.

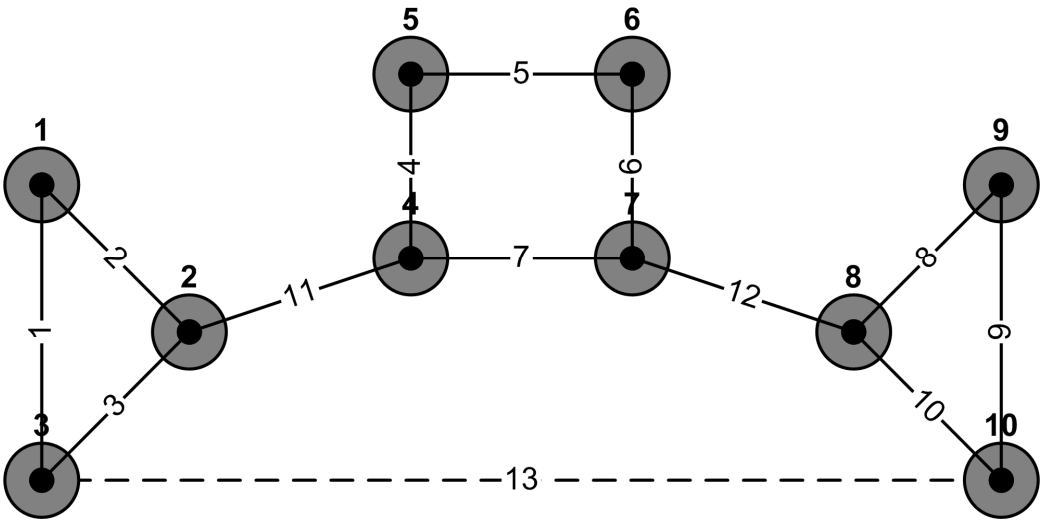

Рис. 19. Двусвязный граф

3.1.3. Построение *ST*-графа

***St*-граф** – это двусвязный ориентированный граф, в котором для выделенного стока *t* (*target*) существуют только входящие ребра, для выделенного источника *s (source)* – только исходящие, а для остальных вершин – обязательно существуют входящие и исходящие ребра. Необходимо в полученном на вход графе переориентировать ребра так, чтобы он стал *st*-графом (*st*-ориентировать дираф).

Эта задача эквивалентна задаче о построении *st*-нумерации. *St*-нумерацией называется такая нумерация вершин, что любая вершина с номером, не равным максимальному или минимальному, имеет инцидентные вершины с большими и меньшими номерами. При этом номера вершин совпадать не могут. Для решения этой задачи применим алгоритм, описанный в работах [24], [25].

Пример *st*-графа приведен на рис. 20.

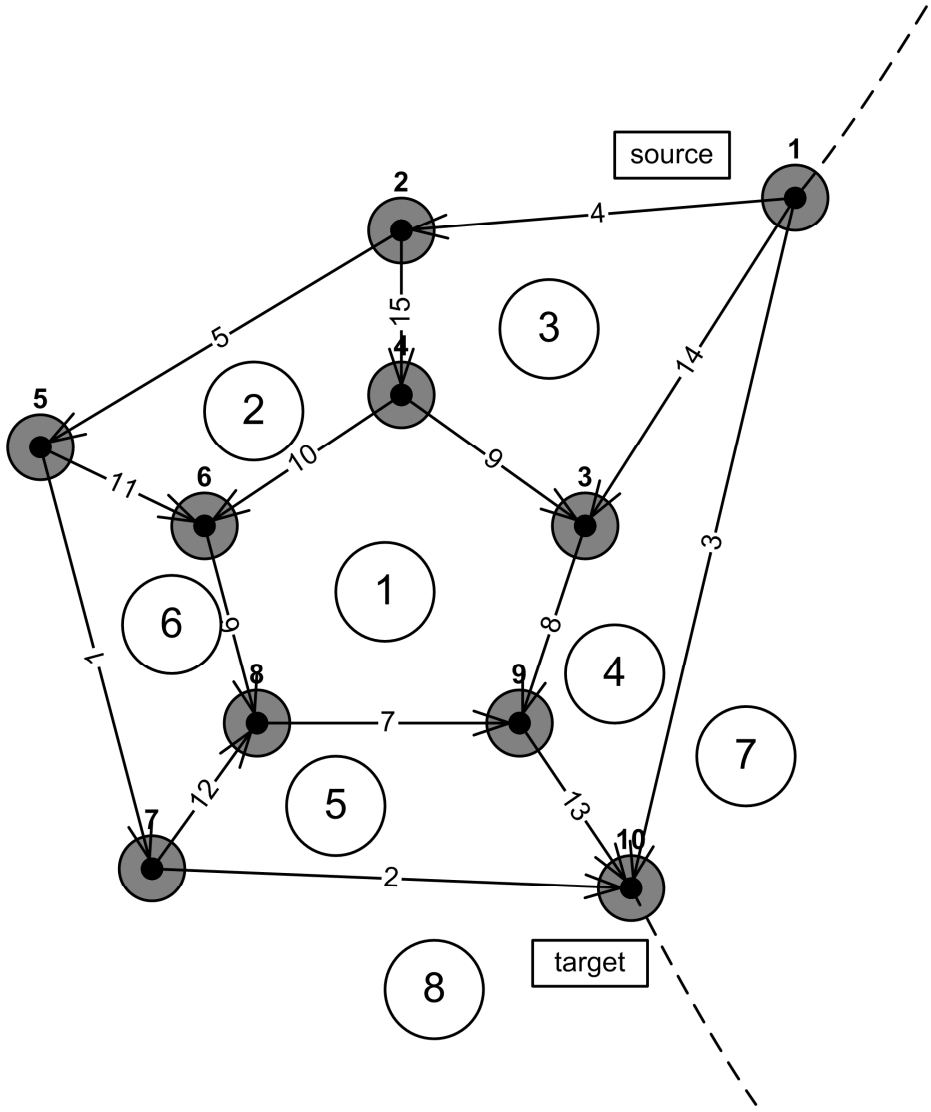

Рис. 20. St-граф

3.2. Алгоритм *GIOTTO*

Алгоритм *GIOTTO* [8] ортогональной укладки диаграмм состояний *UML* является аналитическим. Он был описан и исследован в работе [9].

Краткий план работы алгоритма таков:

1. Замена составных состояний на цепочки вершин – построение графа из диаграммы состояний.
2. Преобразование графа в связный (разд. 3.1.1).

3. Преобразование графа в двусвязный (разд. 3.1.2).
4. Выделение планарного подграфа.
5. Построение *st*-графа (разд. 3.1.3).
6. Построение планарного «надграфа».
7. Разбиение вершин инцидентностью больше четырех на цепочки.
8. Построение ортогонального представления.
9. Минимизация площади, занимаемой уложенным графом ([26, 27]).
10. Удаление фиктивных элементов, созданных в процессе предыдущих шагов.
11. Приведение составных состояний к прямоугольной форме.
12. Восстановление информации об ориентации ребер (информация «теряется» на четвертом шаге).

На рис. 21 приведено ортогональное представление графа с рис. 20.

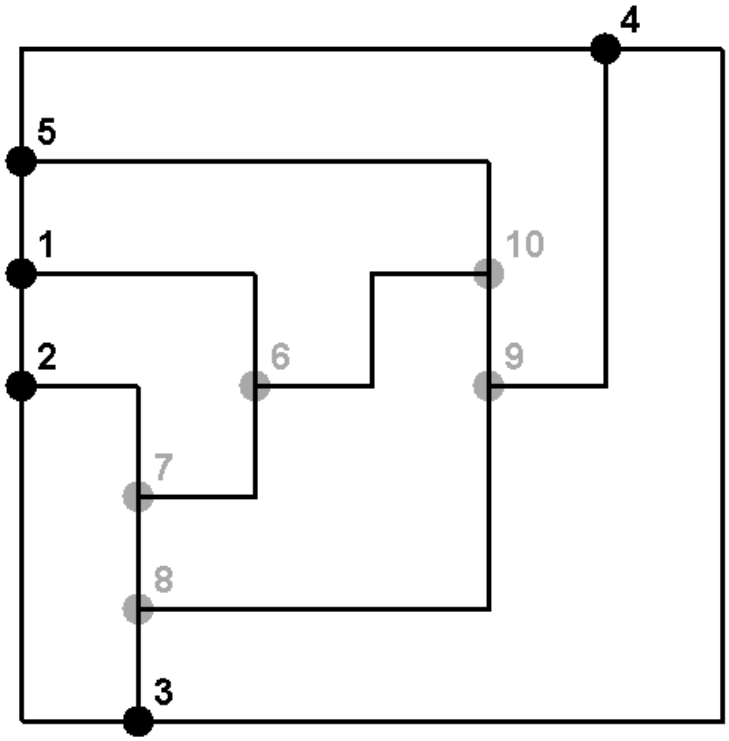

Рис. 21. Ортогональное представление

В работе [8] приведена оценка быстродействия алгоритма построения ортогональной укладки без учета времени, затраченного на выделение граней. Асимптотическая оценка – $O(V^2 logV)$, наиболее сложным этапом является минимизация занимаемой графом площади (именно на этом этапе и достигается такая оценка). На графах среднего размера (порядка 50 вершин) рассмотренные реализации алгоритма ([14 – 16]) работали достаточно быстро.

Об адаптации этого алгоритма к решению задачи укладки диаграммы состояний сказано в разд. 4.2.

3.3. Алгоритм *QMATH-4*

Данный алгоритм необходим для понимания принципов построения алгоритма *QMATH* и, соответственно, *QMATH-STATECHART* (последний реализован в проекте *UniMod*). Похожие алгоритмы можно найти в работах [25] и [28]. Алгоритм осуществляет укладку графов с максимальной степенью вершины равной четырем. Такой выбор позволит визуализировать вершины как точки в ячейках координатной сетки с ребрами по линиям сетки (соответственно, максимум четыре ребра к каждой вершине). В дальнейшем мы будем применять к этой сетке понятия «ряд» и «колонка», по аналогии с рядами и колонками в таблицах. Начнем с элементарной версии алгоритма.

1. Построение диграфа (добавление фиктивных ребер) и *st*-нумерация (граф *st*-ориентируется).
2. Вершины в укладку добавляются в порядке *st*-нумерации.
3. В укладку добавляется вершина *s*, занимая первый (нижний) ряд и для всех исходящих ребер резервируются колонки.
4. На каждом из последующих шагов, в укладку добавляется очередная вершина (в новый ряд и колонку). В момент добавления вершины, мы можем изобразить все входящие ребра, используя уже зарезервированные колонки (по свойству *st*-нумерации все эти ребра идут из вершин с меньшими номерами, уже добавленных в укладку). Кроме того, резервируются колонки для каждого исходящего ребра.

На рис. 23 приведена укладка графа (рис. 22) с помощью элементарной версии алгоритма. Обратим внимание, что каждая вершина всегда занимает ровно один горизонтальный ряд.

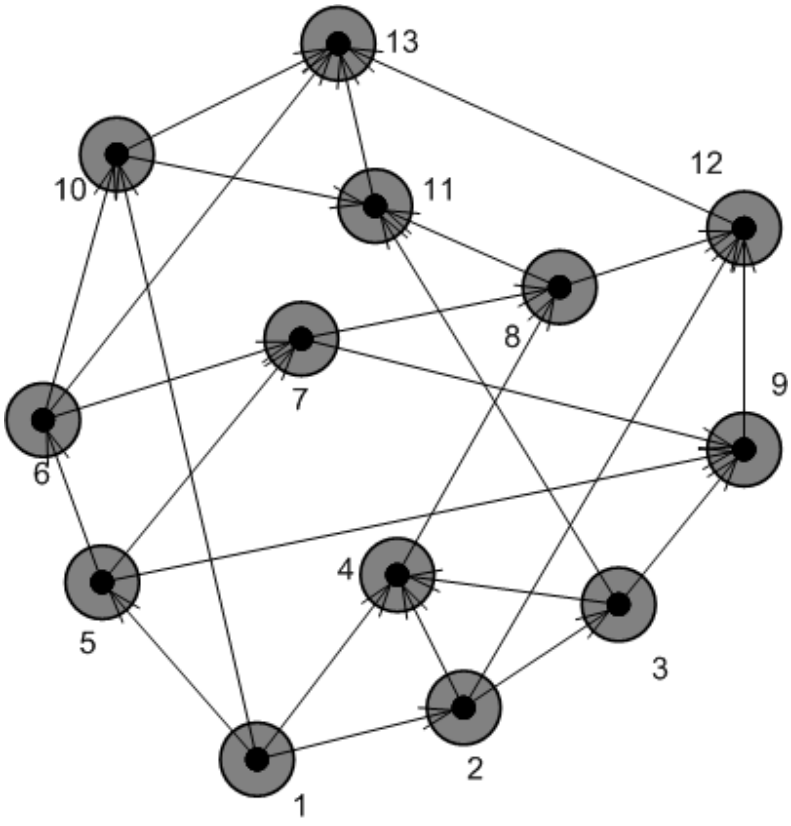

Рис. 22. *ST*-нумерация исходного графа

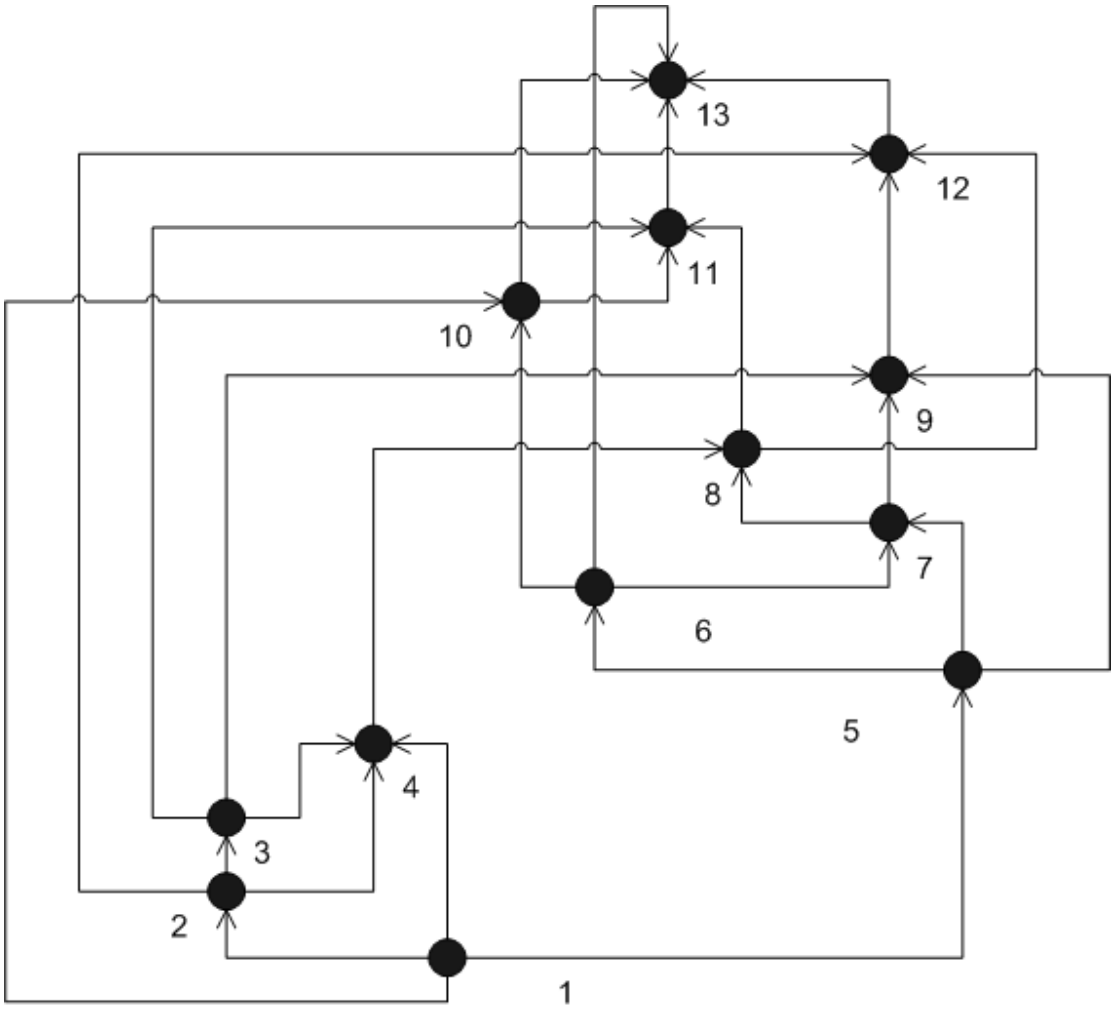

Рис. 23. Укладка базовой версией *QMATH*

3.3.1. Выделение пар вершин

Элементарная версия алгоритма показывает общий подход к укладке, и мы не будем останавливаться на ней подробно. Основной проблемой этого алгоритма является то, что колонки (ряды) используются только один раз. Используем вспомогательные алгоритмы для того, чтобы располагать вершины в одной колонке (одном ряду). Эти алгоритмы строят пары вершин в ряду (колонке).

1. Пара вершин в ряду:
    - Обе вершины находятся в одном ряду в укладке.

- Вершины хотя и находятся в разных рядах, но хотя бы одна из них располагается в одном ряду в какой-либо другой вершиной.

2. Пара вершин в колонке:
   - Вершины расположены в окончательной укладке таким образом, что в одной колоне лежат вертикальные участки как минимум двух ребер.

Пусть G – дираф с максимальной инцидентностью вершины равной четырем. Вершины с a входящими ребрами и b исходящими будем называть *a-b* вершинами (например: *1-1* вершина, *1-2* вершина, *2-2* вершина и т.д.). Упростим граф, исключив из него все *1-1* вершины, исходящее ребро которых является входящим для *1-2* или *1-3* вершины. Эти вершины будут добавлены на заключительном этапе алгоритма. Обозначим упрощенный граф *G'*, а число вершин в нем – *n'*.

Алгоритм формирования пар выглядит следующим образом:

1. На вход алгоритма передается упрощенный граф *G'* и пустой список пар.
2. $i = n' - 1$.
3. Выполняем следующие действия пока i больше двух:
   a. Берем очередную вершину v_i в порядке, обратом порядку *st*-нумерации.
   b. Если v_i вершина является *1-1, 2-1* или *3-1* вершиной, то $i = i - 1$.
   c. Если вершина v_i является *1-2* или *1-3* вершиной, то
      - Добавим в список пар пару (v_{i-1}, v_i).
      - $i = i - 2$.
   d. Если вершина v_i является *2-2* вершиной, то

- Найдем минимальный номер j, такой что, либо v_{i-j} не является *1-1, 2-1* или *3-1* вершиной, либо существует ребро из v_{i-j} в v_i. Если такой номер не существует – $i = i - 1$.
- Если существует:
  1. Добавим в список пар пару (v_{i-j}, v_i).
  2. $i = i - j - 1$.

Рассмотрим правила укладки пар (не будем давать формальное описание правил, так как этот алгоритм не используется в явном виде при укладке состояний). Возьмем пару (v_i, v_j).

1. Если v_i – *2-2* вершина, то возможны следующие случаи:
   a. v_j - *2-2* вершина:
      - Существует ребро из v_j в v_i. Тогда можно использовать колонку дважды, как показано на рис. 24, a.
      - В противном случае, можно использовать одну колонку двумя ребрами, соответствующим образом расположив входящие ребра двух вершин (рис. 24, b и c). В этом случае может оказаться необходимым поменять рядами вершины i и j.
   b. v_j – *1-1, 2-1* или *3-1* вершина и существует ребро из v_j в v_i:
      - Если v_j – *2-1* или *3-1* вершина, то колонка может быть использована дважды, как показано на рис. 24, d.
      - Если же v_j – *1-1* вершина, то можно расположить v_j и v_i в одном ряду, как показано на рис. 24, e.

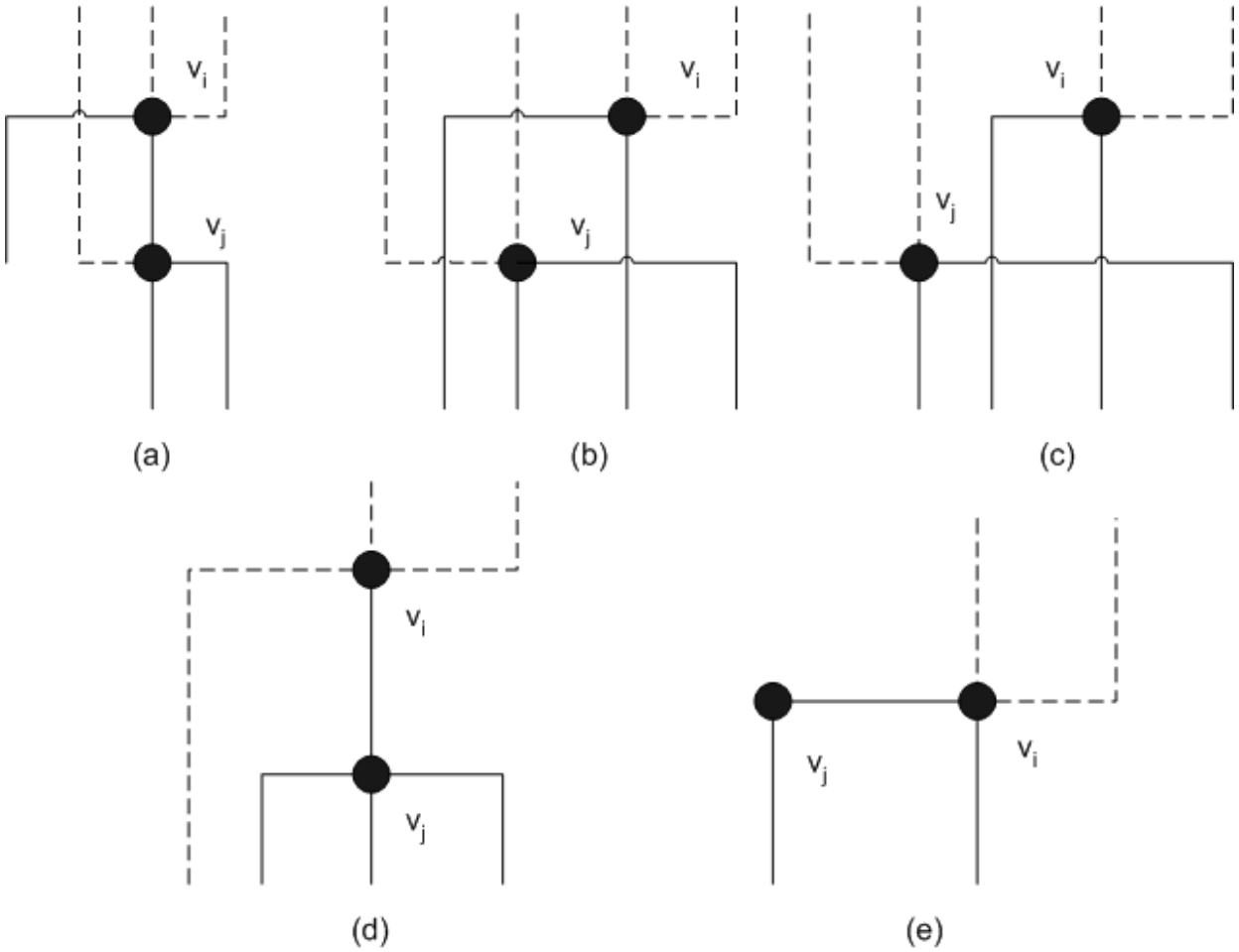

Рис. 24. Расположение 2-2 вершины (1)

c. v_j – *1-2* или *1-3* вершина:

- Существует ребро из v_j в v_i. Тогда вершины могут быть расположены в одном ряду. Точнее, вершина v_i может быть расположена в ряду вершины v_j и в колонке другой вершины, из которой есть входящее ребро в v_i. Заметим, что в соответствии с алгоритмом построения пар, эта вершина v_k имеет номер меньше чем *j* (рис. 25, a).
- В противном случае, можно использовать колонку повторно, как показано на рис. 25, b.

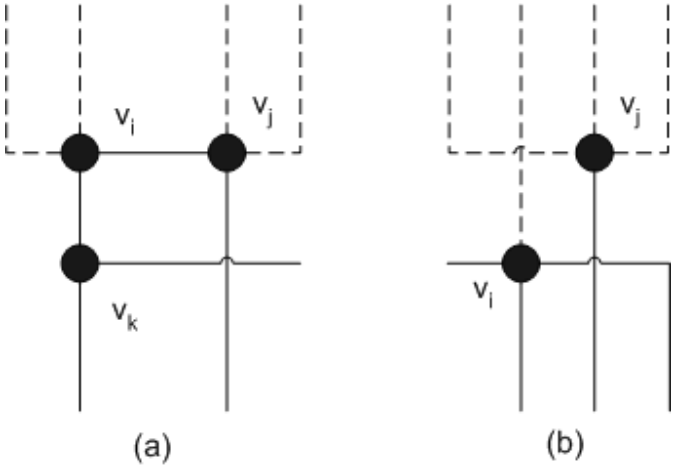

Рис. 25. Расположение 2-2 вершины (2)

2. Если v_i – *1-2* или *1*-3 вершина, то она может быть в паре только с вершиной v_{i-1}. Возможны следующие случаи:

   a. v_{i-1} – *2-2, 2-1* или *3-1* вершина. Колонка используется повторно, так же как и в пунктах 1.a и 1.b.

   b. v_{i-1} – *1-2* или *1-3* вершина и ребра из v_{i-1} в v_i не существует. Тогда вершины могут быть расположены в одном ряду, как показано на рис. 26, а.

   c. v_{i-1} – *1-2* или *1-3* вершина, существует ребро из v_{i-1} в v_i и выполнены одновременно следующие условия:

      - Ребро (v_{i-1}, v_i) не «содержит» не одной *1-1* вершины удаленных при упрощении графа.
      - Выполнено хотя бы одно из следующих условий:
        - Вершина v_i связана с другой вершиной с большим номером (положим, v_j) которая является *1-1, 1-2* или *1-3* вершиной.

- Вершина v_i связана с *2-2* вершиной $v_{j,}$ с большим номером, которая является либо второй вершиной пары вида 1.c (рис. 25, a) или пары вида 1.b (рис. 24, e).

В таком случае, v_i и v_{i-1} можно расположить в одном ряду, как показано на рис. 26, b. Ребро e, соединяющее v_i с v_j будет иметь два излома и будет уложено окончательно при укладке вершины v_j.

d. v_{i-1} – *1-2* или *1-3* вершина, существует ребро из v_{i-1} в v_i и одно из условий предыдущего пункта не выполнено. В этом случае вершины v_{i-1} и v_i располагаются в различных рядах, но один из рядов будет использован дважды позже, как показано на рис. 26, c и d.

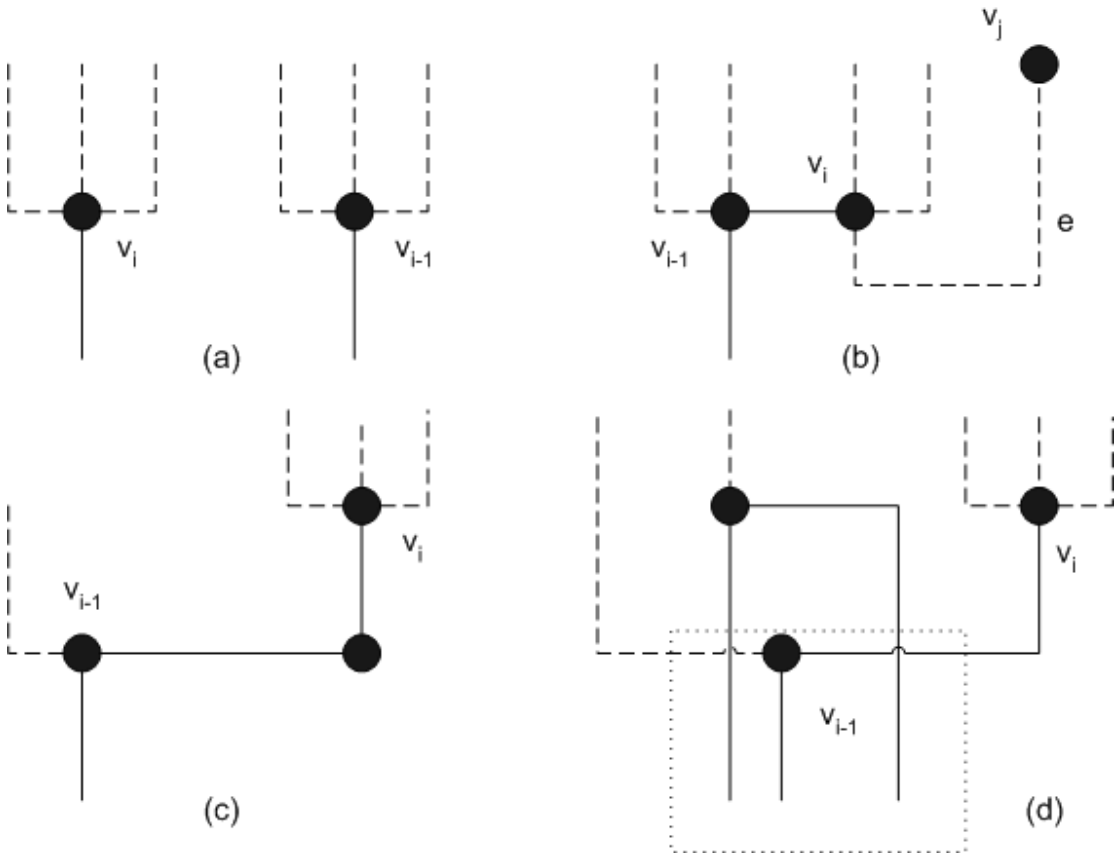

Рис. 26. Расположение других вершин

3.3.2. Полная версия алгоритма

Перейдем к окончательному алгоритму укладки *QMATH-4*.

1. На вход алгоритму передается st-ориентированный дираф.
2. Строится упрощенный граф *G'*.
3. По приведенному выше алгоритму строится список пар.
4. Расположим в укладке первые две вершины (рассматривается отдельно).
    - Если v_2 не принадлежит к паре, в которой она занимает один ряд с какой-либо другой вершиной, то v_1 и v_2 должны занимать один ряд (рис. 27, a).

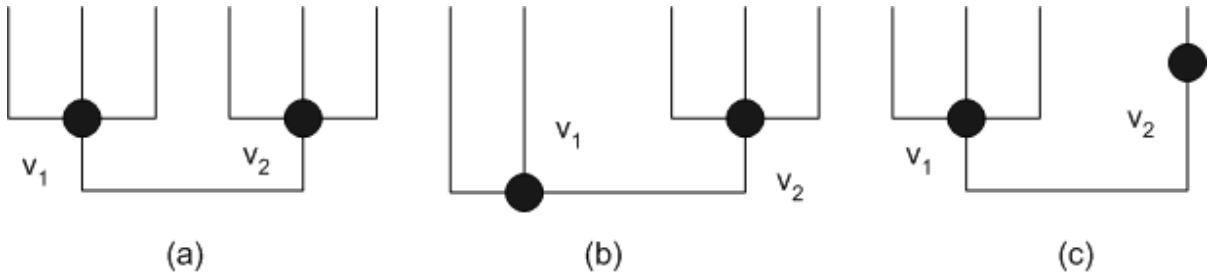

Рис. 27. Расположение первой вершины

    - Если v_1 и/или v_2 имеют инцидентность менее четырех, то для их расположения может потребоваться два ряда. Этот случай показан на рис. 27, b.
    - Если же v_2 принадлежит к паре с другой вершиной, а инцидентность v_1 равна четырем, то вершина v_2 будет уложена, когда алгоритм дойдет до соответствующей пары (рис. 27, c).
5. До тех пор, пока остается более одной вершины (расположение последней вершины – $v_{n'}$ будет рассмотрено отдельно) повторяем следующие шаги
    - Возьмем следующую в порядке *st*-нумерации вершину v_i.

- Если расположение вершины в укладке еще не определено, то:
  - Если вершина не принадлежит ни одной паре, то выделим для нее новый ряд. Уложим все входящие в эту вершину ребра, выделяя при этом новые колонки.
  - Если вершина принадлежит паре, то уложим ее вместе с другой вершиной пары в соответствии с правилами укладки пар.

6. Выделим новый ряд для вершины $v_{n'}$. Если степень этой вершины равна четырем, то ребро, соединяющее $v_{n'}$ с $v_{n'-1}$ будет иметь два излома.
7. Восстановим *1-1* вершины, которые были исключены на этапе упрощения графа следующим образом:
   - в большинстве случаев, эти вершины могут быть расположены в точках изломов ребер;
   - если на ребре нет изломов, то вершина может быть расположена в произвольном узле координатной сетки, в котором нет пересечения ребер;
   - наконец, если таковых на ребре нет, то нужно добавить новый ряд или колонку.

На рис. 22 дана *st*-нумерация исходного графа, а на рис. 28 – укладка графа алгоритмом *QMATH-4*. Обратим внимание на то, что исходный граф является достаточно «сложным» для этого алгоритма – все вершины имеют максимальную степень – четыре.

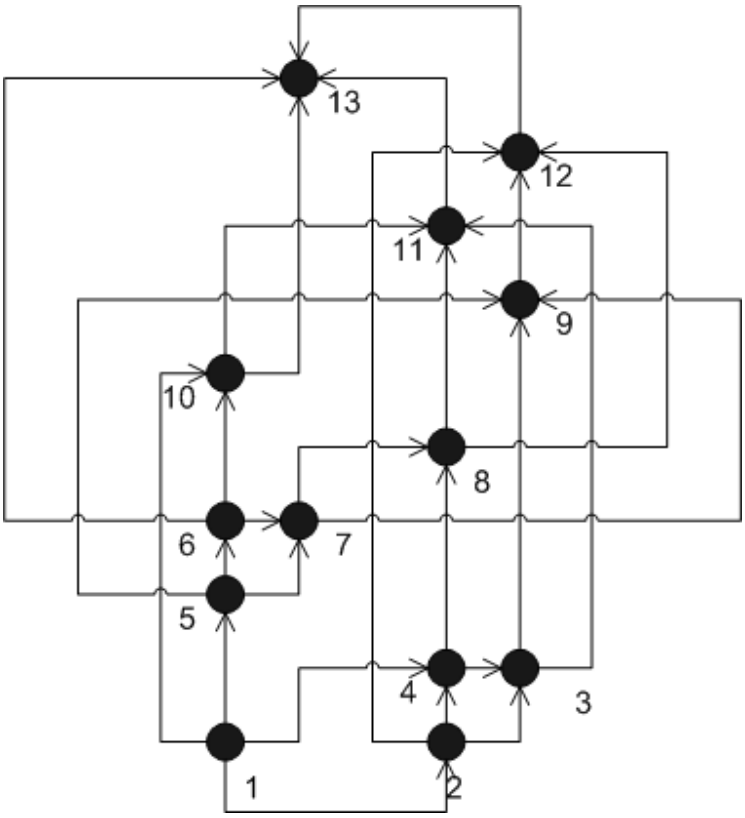

Рис. 28. Укладка алгоритмом *QMATH-4*

4. РАЗРАБОТАННЫЕ АЛГОРИТМЫ

Были разработаны следующие алгоритмы:

- Метод отжига с ортогонализацией;
- *QMATH;*
- *QMATH-STATECHART.*

Также была изучена возможность адаптации алгоритма *GIOTTO* к задаче укладки диаграмм состояний.

4.1. Метод отжига с ортогонализацией

В рамках работы ([9]) был разработан и реализован алгоритм укладки диаграмм состояний методом отжига с применением ортогонализации. Развитие этого метода было продолжено в настоящей работе. Проходя по состояниям, алгоритм пытается сдвинуть каждое из них. Если производится сдвиг простого состояния, вложенного в составное состояние, то запрещается выход за пределы объемлющего составного состояния. После сдвига каждого состояния вычисляется новое значение штрафной функции. Если штрафная функция уменьшилась – новое расположение вершин фиксируется и производится сдвиг следующего состояния, если увеличилась – перемещение вершины отклоняется. Колебания системы затухают – расстояния, на которые сдвигаются состояния, уменьшаются на каждом шаге. Отдельным шагом выполняется ортогонализация укладки. Приведем краткое описание алгоритма.

4.1.1. Построение штрафной функции

Для описания штрафной функции определим **диаметр графа** (D) – максимальный из его линейных размеров. У вершины на диаграмме задан размер по умолчанию (ширина и высота вершины). Обозначим максимальный из линейных размеров вершины по умолчанию как S_m. Каждая вершины при этом, имеет размер,

возможно, превышающий размер по умолчанию. Большой размер имеют как вершины с большой инцидентностью, так и составные вершины (вершины, соответствующие составным состояниям).

В алгоритме используется вспомогательная координатная сетка с крупными ячейками (порядка минимального размера вершины). Вершины располагаются в точках, имеющих целочисленные координаты в крупной сетке (это улучшает качество диаграммы). Таким образом, эстетика выравнивания соблюдается автоматически.

Штрафная функция составлена из слагаемых, приведенных в табл. 1.

Таблица 1. Слагаемые штрафной функции

Название	Метод вычисления слагаемого
Пересечение переходов	Каждое пересечение переходов учитывается с весом D*INTERSECT_MODIFIER. Наложение переходов также считается пересечением.
Отклонение от оптимальной длины перехода	Отклонение длины перехода от заданной оптимальной длины (OPTIMAL_LENGTH) учитывается с весом Δ*LONGER_MODIFIER в случае превышения оптимальной длины и с весом Δ*SHORTER_MODIFIER в противном случае.
Пересечение вершин	Наложение вершин учитывается с весом D*OVERLAY_MODIFIER.
Отклонение от оптимального расстояния между вершинами	Отклонение расстояния между вершинами от заданного оптимального (OPTIMAL_DISTANCE) учитывается с весом Δ*FARTHER_MODIFIER в случае превышения и с весом Δ*CLOSER_MODIFIER в противном случае.

Компактность диаграммы	Превышение расстояния до центра масс графа над корнем из количества вершин ($\sqrt{N}$) учитывается с весом $\Delta * \sqrt{N} * S_m$ *CENTER_MODIFIER.
Прохождение перехода по вершине	Прохождение перехода по вершине учитывается с весом D*OL_MODIFIER*Angle. Так как в процессе ортогонализации переходы, которые исходно являлись наклонными, будут состоять из вертикальных и горизонтальных отрезков, то учитывается отклонение от вертикали или горизонтали (Angle) – нормированное на единицу угловое расстояние от наиболее близкой из координатных осей. Таким образом, предотвращается прохождение горизонтальных и вертикальных переходов по вершинам.

Приведем связь между слагаемыми и эстетиками:

Таблица 2. Соответствие слагаемых и эстетик

Название	**Эстетика**
Пересечение переходов	Минимизация пересечений ребер
Отклонение от оптимальной длины перехода	Ограничение "свободного места"
Пересечение вершин	Минимизация наложения вершин
Отклонение от оптимального расстояния между вершинами	Ограничение "свободного места", минимизация площади
Компактность диаграммы	Минимизация площади

Прохождение перехода по вершине	Минимизация прохождения ребер по вершинам

Зафиксируем оптимальные расстояния между вершинами и длины ребра.

Таблица 3. Значения оптимальных расстояний

Константа	Значение
OPTIMAL_DISTANCE	50
OPTIMAL_LENGTH	50

В ходе вычислительного эксперимента (производилась укладка диаграмм состояний, взятых из ряда проектов, в которых применялся пакет *UniMod*) были выбраны значения множителей при компонентах штрафной функции, приведенные в табл. 4. Эксперимент проводился следующим образом: производилась укладка диаграммы, взятой из проекта, в котором применялся пакет *UniMod*, затем визуально оценивался результат укладки и, в случае если результат оказывался неудовлетворительным, изучалось влияние различных констант, а затем производились соответствующие изменения их значений.

Таблица 4. Значения констант

Константа	Значение
INTERSECT_MODIFIER	1
OL_MODIFIER	5
OVERLAY_MODIFIER	50

SHORTER_MODIFIER	2
LONGER_MODIFIER	1
CLOSER_MODIFIER	0.5
FARTHER_MODIFIER	0.25
CENTER_MODIFIER	1
OPTIMAL_DISTANCE	50
OPTIMAL_LENGTH	50

С помощью оптимизации штрафной функции строим изначальную укладку (в ней ребра являются отрезками), а затем ортогонализуем ее (вообще говоря, с помощью аналитического алгоритма). При этом в процессе ортогонализации могут появиться дополнительные пересечения ребер, не учтенные алгоритмом отжига, но их количество, как показывает практика, невелико.

4.1.2. Ортогонализация укладки

После нахождения некоторого расположения вершин с помощью метода отжига ребра укладываются так, чтобы каждое состояло только из горизонтальных и вертикальных отрезков. В укладке, которая строится в настоящей работе, ребро будет иметь не более одного излома. Алгоритм ортогонализации разработан автором и базируется на идеях, изложенных в [7, 8, 10].

Для нахождения окончательного расположения ребер автором разработан оригинальный алгоритм ортогонализации, состоящий из следующих шагов:

- распределение ребер по сторонам вершин;
- распространение петель по сторонам вершин;
- выделение **портов** (точек входа ребер в вершину) на сторонах вершин.

Опишем предложенный алгоритм. На вход алгоритм получает граф, в котором уже известны геометрические координаты вершин, но не известны ни координаты начала и конца каждого ребра (представлением вершины является прямоугольник, поэтому соответствующие координаты невозможно получить автоматически), ни координаты изломов на ребрах. На выходе алгоритма – укладка графа.

4.1.3. Распределение ребер по сторонам вершин

Если две вершины можно соединить горизонтальным или вертикальным отрезком, то в окончательной укладке их будет соединять прямое ребро (стороны, в которые оно входит, определяются однозначно). В противном случае необходимо выбрать «маршрут» для ребра. Для каждого (непрямого) ребра возможно не более двух вариантов его проведения (маршрутов), как показано на рис. 29 (именно ради упрощения этой части алгоритма было введено ограничение на количество изломов на ребре).

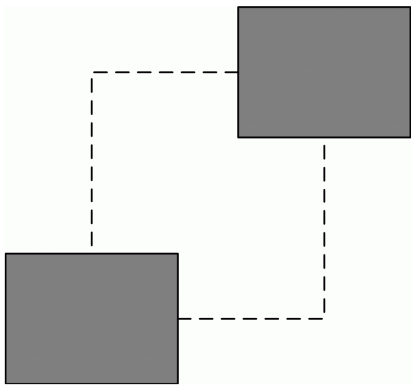

Рис. 29. Варианты проведения ребра

Будем действовать следующим образом:

- определим ребра, для которых известно, какой стороне они принадлежат (какой стороне принадлежит соответствующий порт). Это ребра, маршрут которых уже определен при обработке других вершин, и ребра, соединяющие вершины, лежащие на одной линии;

- определим текущую инцидентность каждой стороны (количество ребер, входящих в эту сторону или исходящих из нее);
- переберем все ребра, для которых есть два варианта их расположения (рис. 29). Каждое из них проводим с той стороны вершины, для которой текущая инцидентность минимальна.

Применим описанный алгоритм к вершинам (обходя их в порядке убывания инцидентности). Таким образом, для каждой вершины определим, с какой ее стороны находятся точки выхода ребер.

Распределим петли по сторонам с минимальной набранной инцидентностью (петлю можно поместить на любой стороне).

4.1.4. Выделение портов

Теперь отсортируем ребра, инцидентные каждой из сторон вершины так, чтобы, по возможности, уменьшить количество пересечений. Рассмотрим множество таких ребер, инцидентных левой стороне вершины *A*. В нижней части рассматриваемой стороны вершины будут расположены порты ребер, ответные вершины которых располагаются ниже вершины *A*. Затем – порты горизонтальных ребер (разд. 4.1.3), а затем – порты таких ребер, что их ответные вершины находятся слева вверху от вершины *A*.

Рассмотрим последнюю группу ребер. Чем ближе по горизонтали находится ответная вершина к вершине *A*, тем выше будет расположен соответствующий порт. Для других относительных расположений рассматриваемой вершины и ответной вершины алгоритм аналогичен (рис. 30).

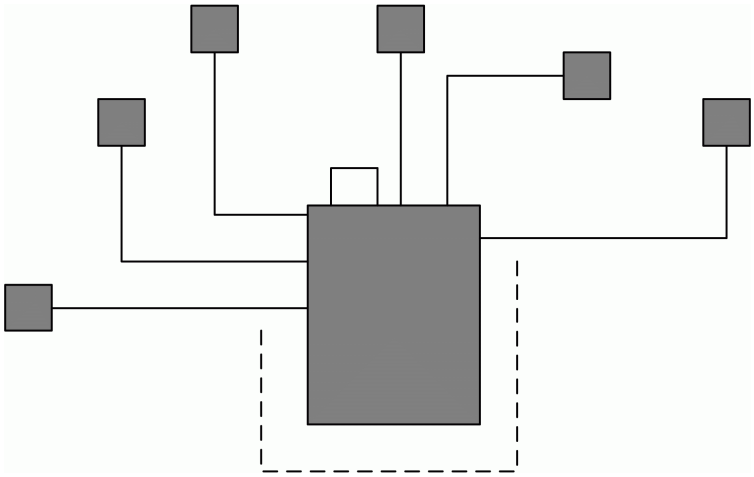

Рис. 30. Сортировка ребер.

4.1.5. Внедрение

В рамках проекта *UniMod* автором реализован изложенный выше метод отжига с применением ортогонализации. Эта реализация вошла в *UniMod Release 1 Build 07*. Создание данного алгоритма позволило обеспечить загрузку файлов старых форматов (созданных без использования графического редактора). Затем от данного алгоритма отказались в пользу аналитических алгоритмов. Причины отказа изложены в разд. 0.

На рис. 32 приведена диаграмма состояний, полученная из исходной диаграммы на рис. 31 с помощью алгоритма отжига с последующей ортогонализацией, а на рис. 33 та же диаграмма, преобразованная вручную.

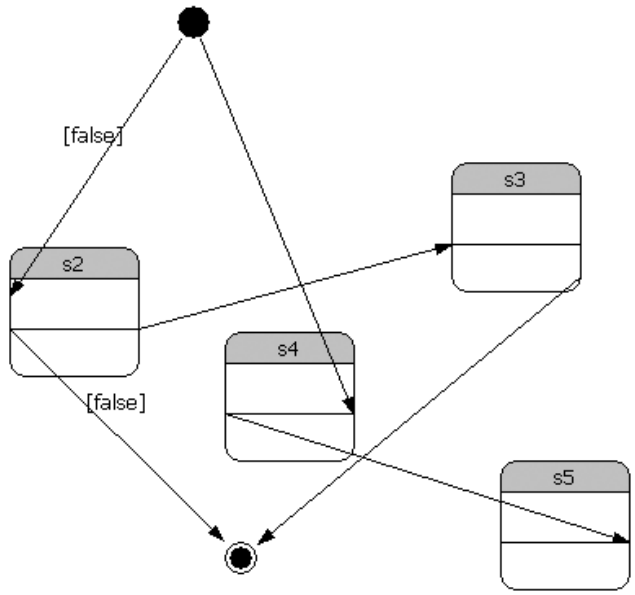

Рис. 31. Исходная диаграмма

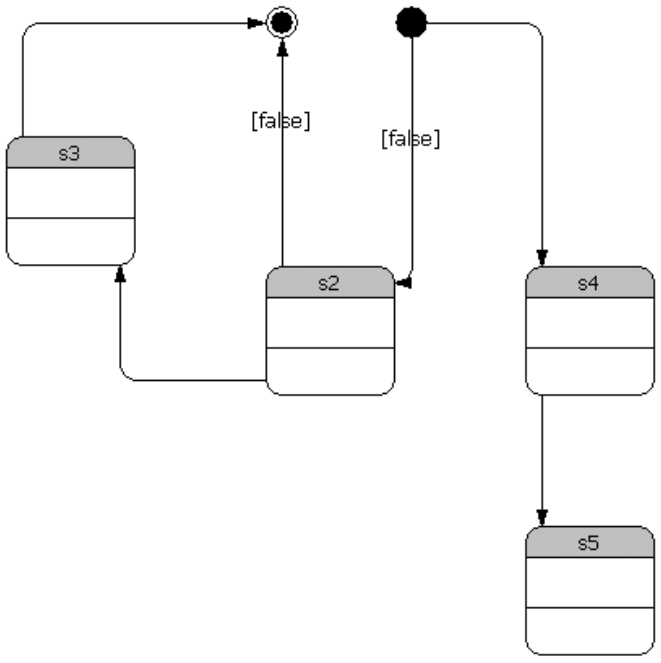

Рис. 32. Уложенная диаграмма

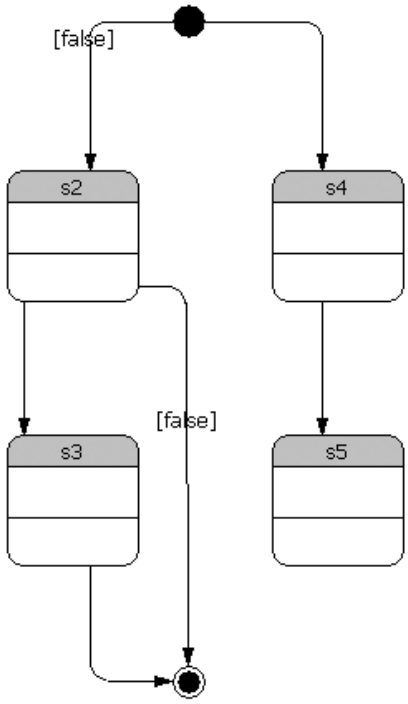

Рис. 33. Диаграмма, преобразованная вручную

ВЫВОДЫ ПО РАЗДЕЛУ 4.1

Штрафная функция вычисляется за время асимптотически порядка $O(E^2)$, где E – количество ребер в графе (константа при оценке велика, так как последовательно вычисляется несколько различных слагаемых штрафной функции). Для графов среднего размера (порядка 50 вершин) граф, как показала экспериментальная проверка, должен испытать количество возмущений приблизительно равное количеству вершин для достижения приемлемого качества укладки, таким образом, время работы алгоритма для таких графов - $O(VE^2)$, при этом с большой константой при асимптотической оценке. На больших графах время работы алгоритма неудовлетворительно.

Другая проблема заключается в сильной неопределенности результата (выражается в том, что качество укладки значительно изменяется от запуска к запуску алгоритма). Последнее обстоятельство вызывается наличием у штрафной функции большого количества локальных минимумов.

Из изложенного следует (и этот результат подтверждается экспериментально), что данный алгоритм, основанный на методе отжига, хорошо работает лишь на несложных диаграммах, с ростом же количества вершин снижается качество укладки (вследствие попадания штрафной функции в окрестность одного из локальных минимумов) и увеличивается время работы алгоритма.

Исследование алгоритмов, используемых в существующих продуктах, и сравнение результатов также приводит нас к выводу, что метод отжига для решения данной задачи менее предпочтителен, чем аналитические алгоритмы. Обратим также внимание, что даже реализованный алгоритм является смешанным: ортогонализация укладки и обеспечение выравнивания вершин производятся аналитически. Перейдем теперь к аналитическим алгоритмам.

4.2. Модифицированный алгоритм *GIOTTO*

Проводились исследования, посвященные адаптации алгоритма укладки *GIOTTO* к решению задачи укладки диаграмм состояний. К сожалению, не удалось создать модификации алгоритма *GIOTTO* для укладки диаграмм состояний без ухудшения качества окончательной укладки, что связано с тем, что алгоритм производит сложные вспомогательные преобразования (использует другие представления графа – ортогональное представление и представление видимости). В известной автору литературе упоминания о таких модификациях также не встречаются.

Дело в том, что после приведения составных состояний к прямоугольной форме, размер этих состояний делается неоправданно большим и необходимо разработать и использовать дополнительные аналитические алгоритмы для того, чтобы минимизировать их размер и провести ребра, ведущие в составные состояния.

Рассмотрим представление видимости графа (рис. 34), в котором цепочка вершин *3–8–9–10–4–12* (ребра выделены пунктиром) соответствует составному состоянию. На рис. 35 произведен сдвиг элементов диаграммы и построен объемлющий прямоугольник для составного состояния, который занимает неоправданно много места.

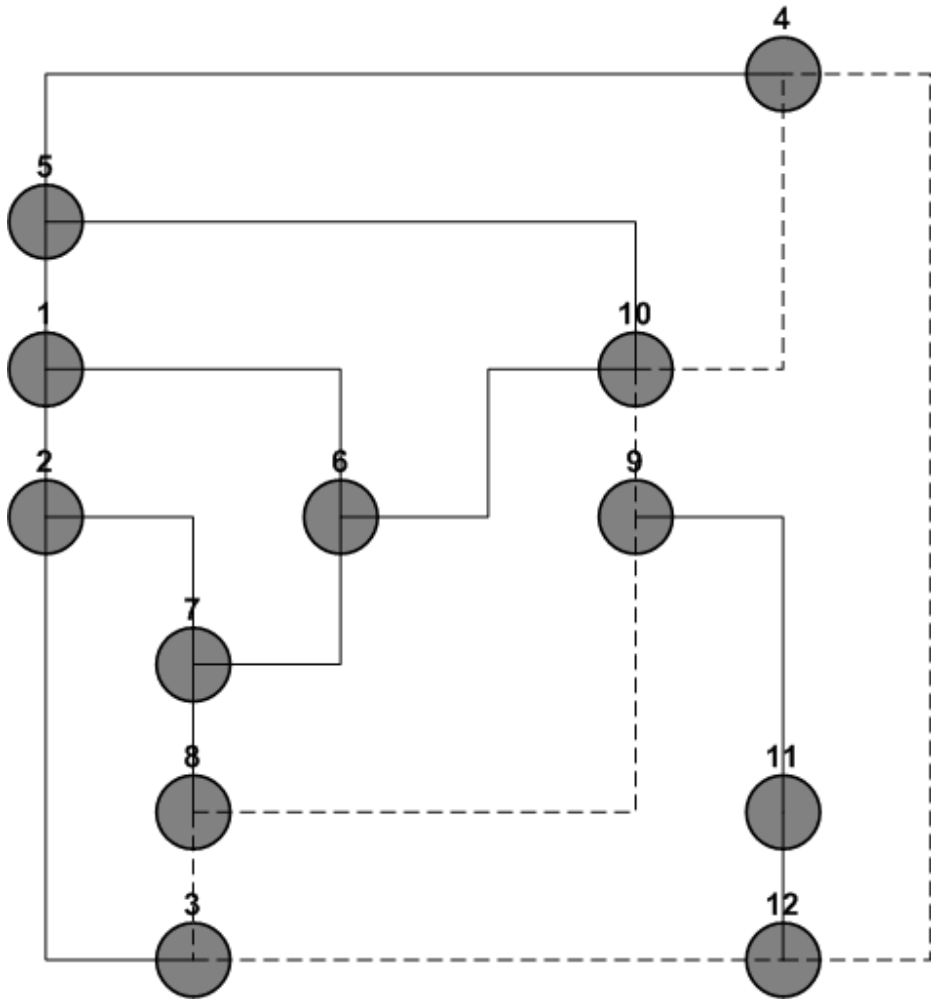

Рис. 34. Пример составного состояния

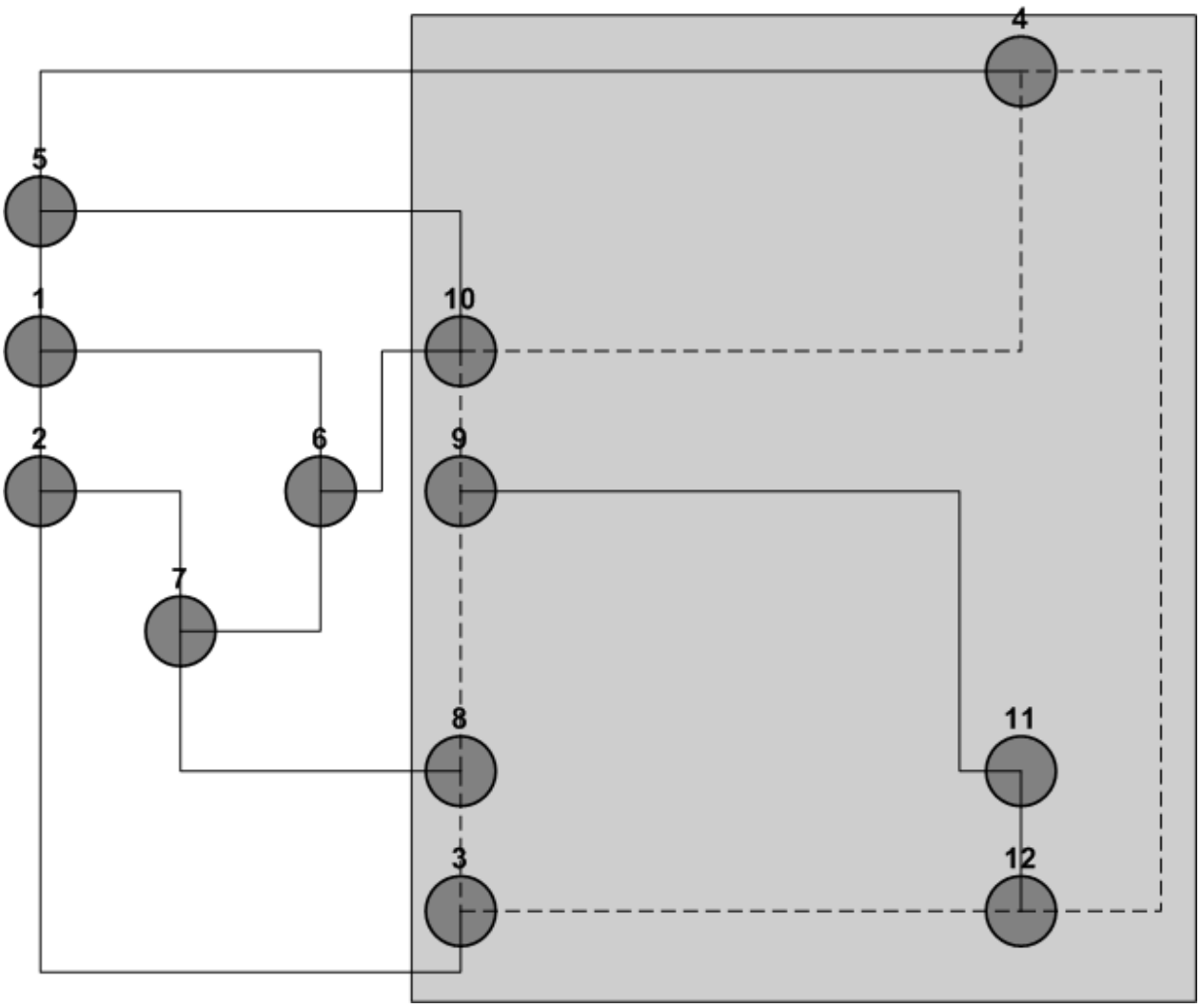

Рис. 35. Замена на объемлющий прямоугольник

Более сложные изменения алгоритма для адаптации к диаграмме состояний, в свою очередь, не позволят сохранить качество результатов *GIOTTO* в остальной части. В итоге, после сравнения данного алгоритма с более простыми (с точнее зрения реализации и адаптации) алгоритмами ортогональной укладки был разработан алгоритм *QMATH-STATECHART*, о котором будет рассказано ниже.

4.3. Алгоритм *QMATH*

Разработанный автором алгоритм *QMATH* укладки графов является аналитическим; основными идеями он опирается на алгоритм *QMATH-4* и является базой для алгоритма укладки диаграмм состояний *QMATH-STATECHART*. В *QMATH* нет ограничений на степень вершины.

В отличие от алгоритма *QMATH-4*, в котором в качестве представления вершины была выбрана точка (окружность малого радиуса, имеющая центр в узле координатной сетки), здесь в качестве представления вершины выберем прямоугольник. Ребра, инцидентные вершине, могут входить в одну из фиксированных точек на грани прямоугольника (порт). На рис. 36 приведены два варианта представления вершины в виде прямоугольника. Вариант слева позволяет использовать угловой порт для двух ребер, а в варианте справа угловой порт отсутствует. Хотя в проекте *UniMod* использовано представление, показанное на рисунке справа, при работе над алгоритмом будет использоваться представление, показанное слева, как более удобное для исследования (порты находятся в узлах координатной сетки). При переходе к действительно используемому представлению размеры состояний (вершин) могут оказаться на единицу больше.

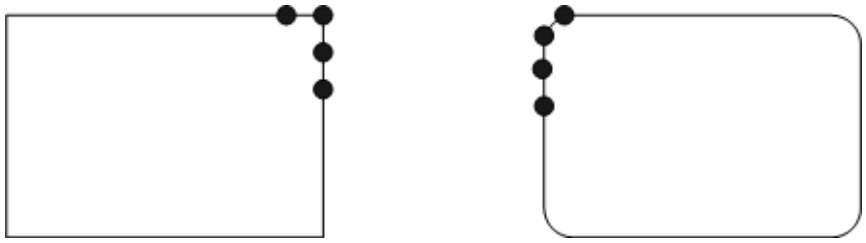

Рис. 36. Варианты представления вершины

Алгоритм *QMATH* работает только с *st*-ориентированными двусвязными графами (вопросы преобразования графа к диграфу, *st*-ориентации и обратного преобразования обсуждались выше). Все ребра направлены от вершин с меньшим номером к вершинам с большим номером. Размер прямоугольника,

соответствующего вершине, определяется в момент добавления вершины в укладку. Все исходящие ребра выходят из портов на верхней грани вершины, все входящие ребра – входят в порты на боковых сторонах. При этом мы не используем точечные вершины (таким образом, минимальный размер вершины – единица).

Рассматривая очередную вершину, распределим входящие ребра между правой и левой гранями прямоугольника. Пусть в вершину входит *in(v)* ребер, тогда $\left\lfloor \frac{in(v)}{2} \right\rfloor$ ребер входят в правую сторону, а оставшиеся $\left\lceil \frac{in(v)}{2} \right\rceil$ – в левую. Таким образом, при добавлении вершины нам потребуется добавить не более $\left\lceil \frac{in(v)}{2} \right\rceil$ рядов и *out(v)* колонок (исключением являются вершины с одним входящим или исходящим ребром, так как для них добавляется две колонки или ряда, соответственно).

Рассмотрим вершину *v* и все вершины u такие, что существует ребро *e = (u, v)*. Пусть число *in(v)* – четное. Тогда найдутся колонки c_1 и c_2, соответствующие ребрам e_1 и e_2, такие что:

- c_1 находится левее c_2;
- есть ровно $\frac{in(v)}{2} - 1$ колонок, содержащих ребра входящие в *v*, и расположенных левее c_1;
- есть ровно $\frac{in(v)}{2} - 1$ колонок, содержащих ребра входящие в *v*, и расположенных правее c_2.

Ребра e_1 и e_2 назовем входящими медианами вершины *v*. e_1 – левая медиана, e_2 – правая медиана.

Если же число *in(v)* – нечетное, то есть ровно одно ребро-медиана e, справа и слева от колонки соответствующей которому есть $\left\lfloor \frac{in(v)}{2} \right\rfloor$ колонок с ребрами, входящими в *v*.

При добавлении вершины в укладку добавляется необходимое количество колонок и рядов. Причем, колонки добавляются между входящими медианами (если их две) или справа от медианы (если она одна).

Для уменьшения занимаемой площади, к алгоритму может быть применена техника использования рядов и колонок, аналогичная таковой в алгоритме *QMATH-4* (подробно эта адаптация описана в [25]). На рис. 38 приведена укладка графа (с рис. 37) алгоритмом *QMATH*.

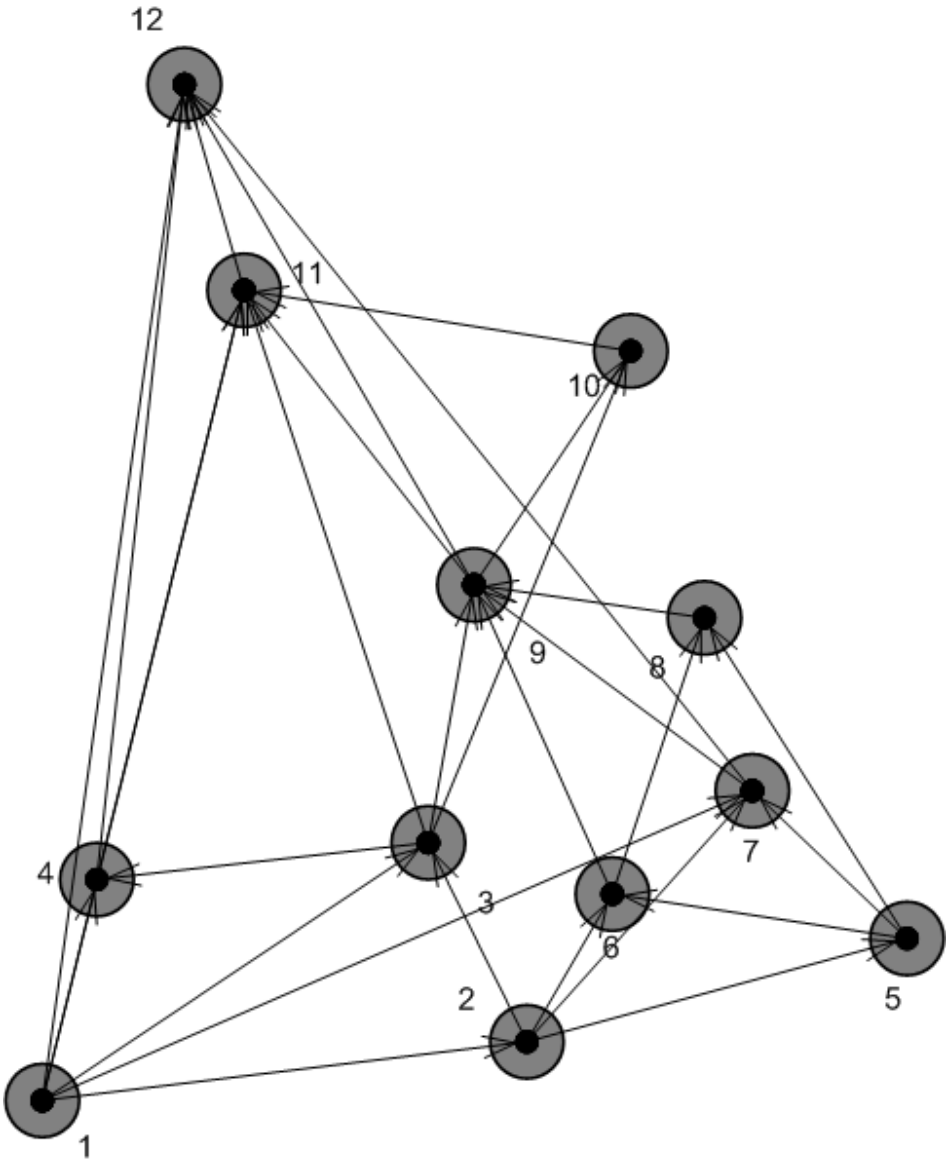

Рис. 37. Исходный граф для *QMATH*

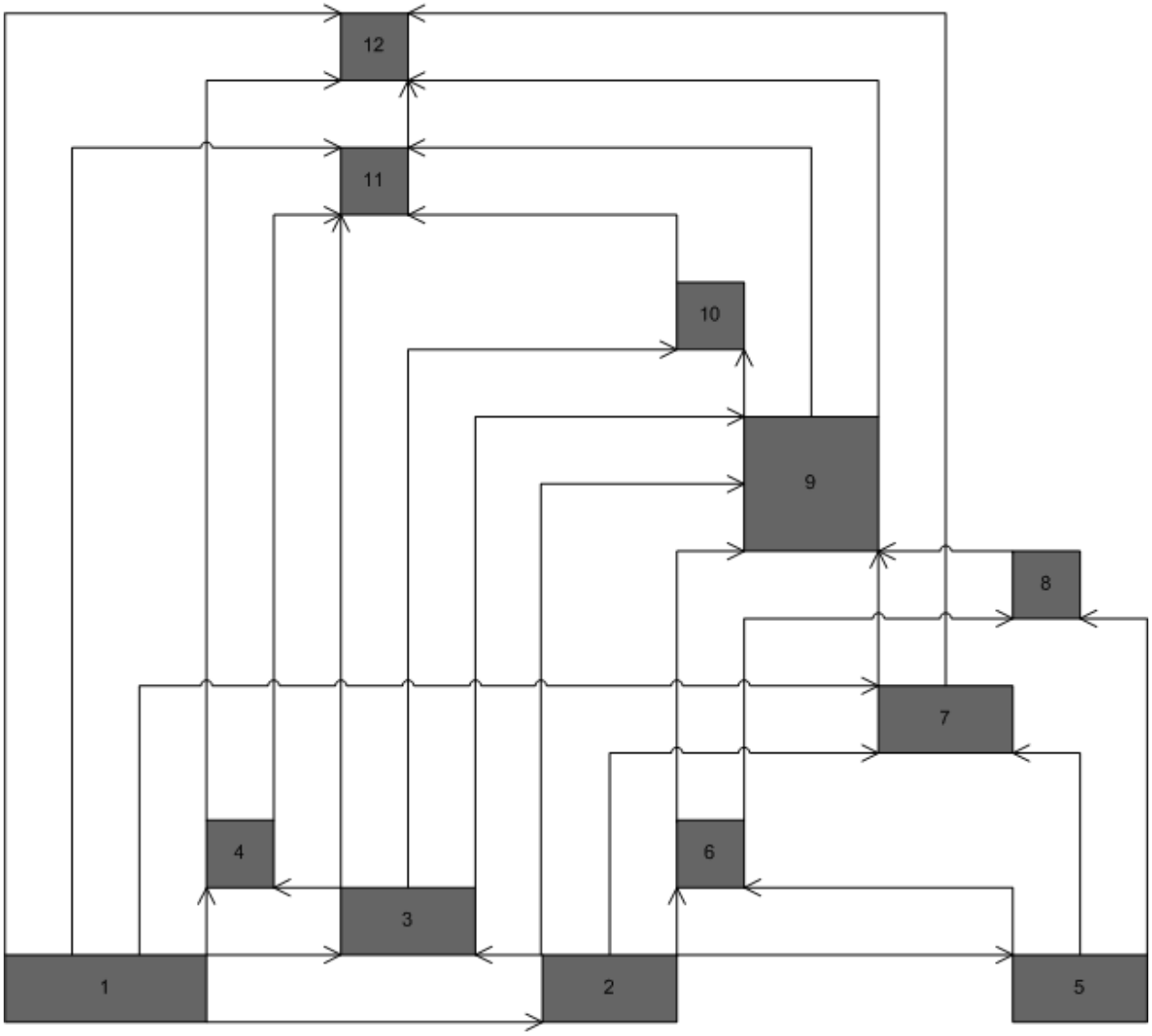

Рис. 38. Укладка графа алгоритмом *QMATH*

4.4. Алгоритм *QMATH-STATECHART*

Очередной разработанный автором алгоритм использует в качестве основы алгоритм *QMATH* и адаптирован для укладки диаграмм состояний. Алгоритм состоит из следующих шагов:

1. Разделение графа на слои.
2. Укладка подграфов в каждом слое.
3. Объединение графов.

4.4.1. Разделение графа на слои

Основная проблема при укладке диаграммы состояний, которая выделяет задачу укладки диаграмм состояний в отдельную подзадачу – это проблема укладки составных состояний. В рамках рассматриваемого алгоритма мы, на первом шаге, будем рассматривать множества состояний, находящихся непосредственно в одном составном состоянии, отдельно. Таким образом, будет получен набор графов, которые будут подвергнуты укладке независимо друг от друга. Для ребер, соединяющих состояния из различных подграфов, будут зарезервированы порты (это необходимо при определении размеров состояний). При этом необходимо учитывать *st*-нумерацию на каждом слое (чтобы определить, будут ли ребра, которые необходимо добавить позже, входящими или исходящими) На рис. 39 приведен пример укладки графа, в вершине *v5* которого расположен вложенный слой.

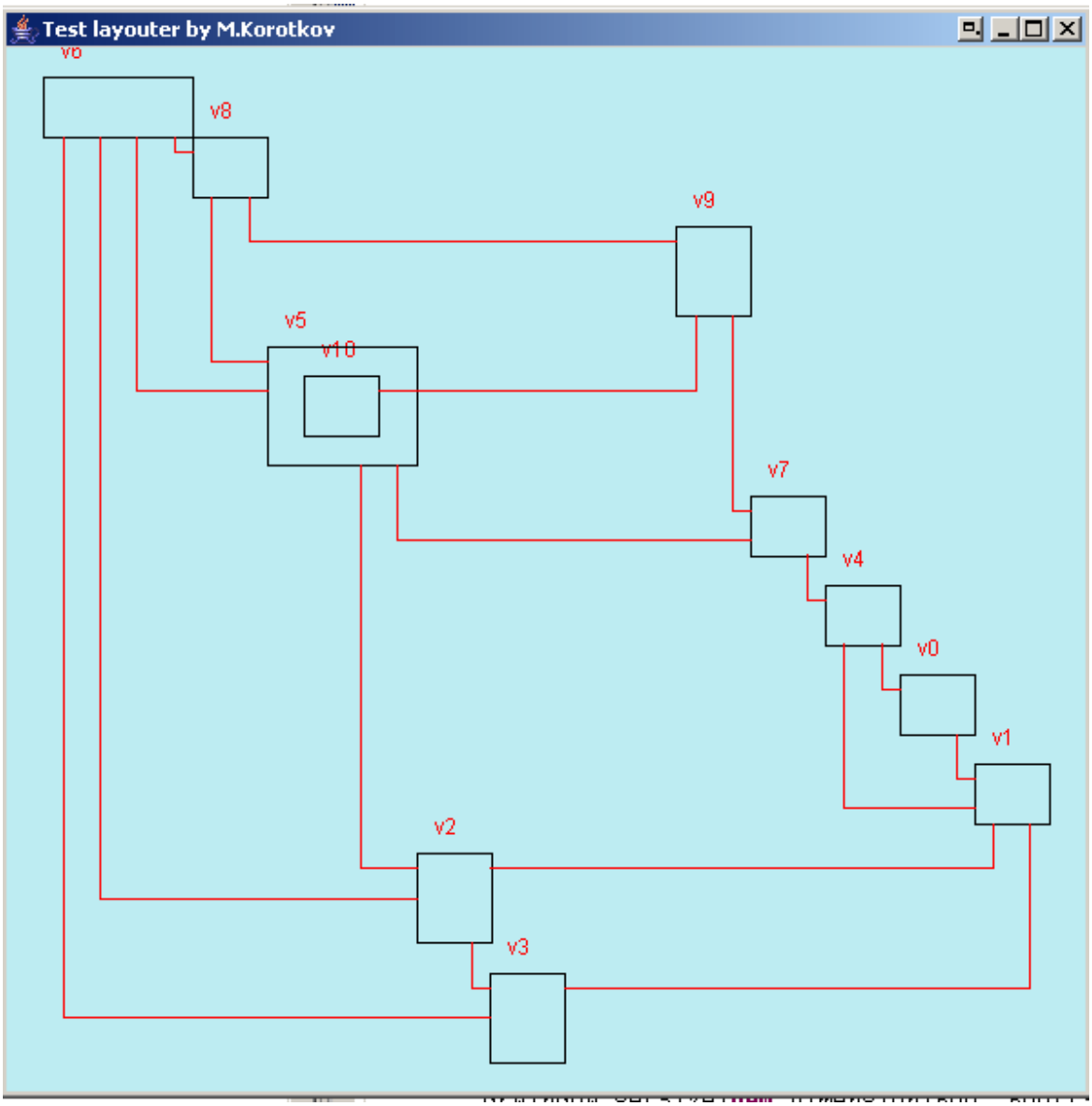

Рис. 39. Укладка графа со слоями

4.4.2. Укладка подграфов

Каждый подграф, полученный на предыдущем шаге, укладывается с помощью алгоритма *QMATH*. При определении размеров состояний учитывается информация о входящих и исходящих ребрах, которые соединяют вершины различных подграфов.

4.4.3. Объединение графов

На этапе объединения графов добавляются ребра между слоями, а также рассчитываются окончательные размеры состояний, в которых есть вложенные. На рис. 40 приведен пример укладки многослойного графа, соответствующего диаграмме состояний, в которой есть состояния второго уровня вложенности.

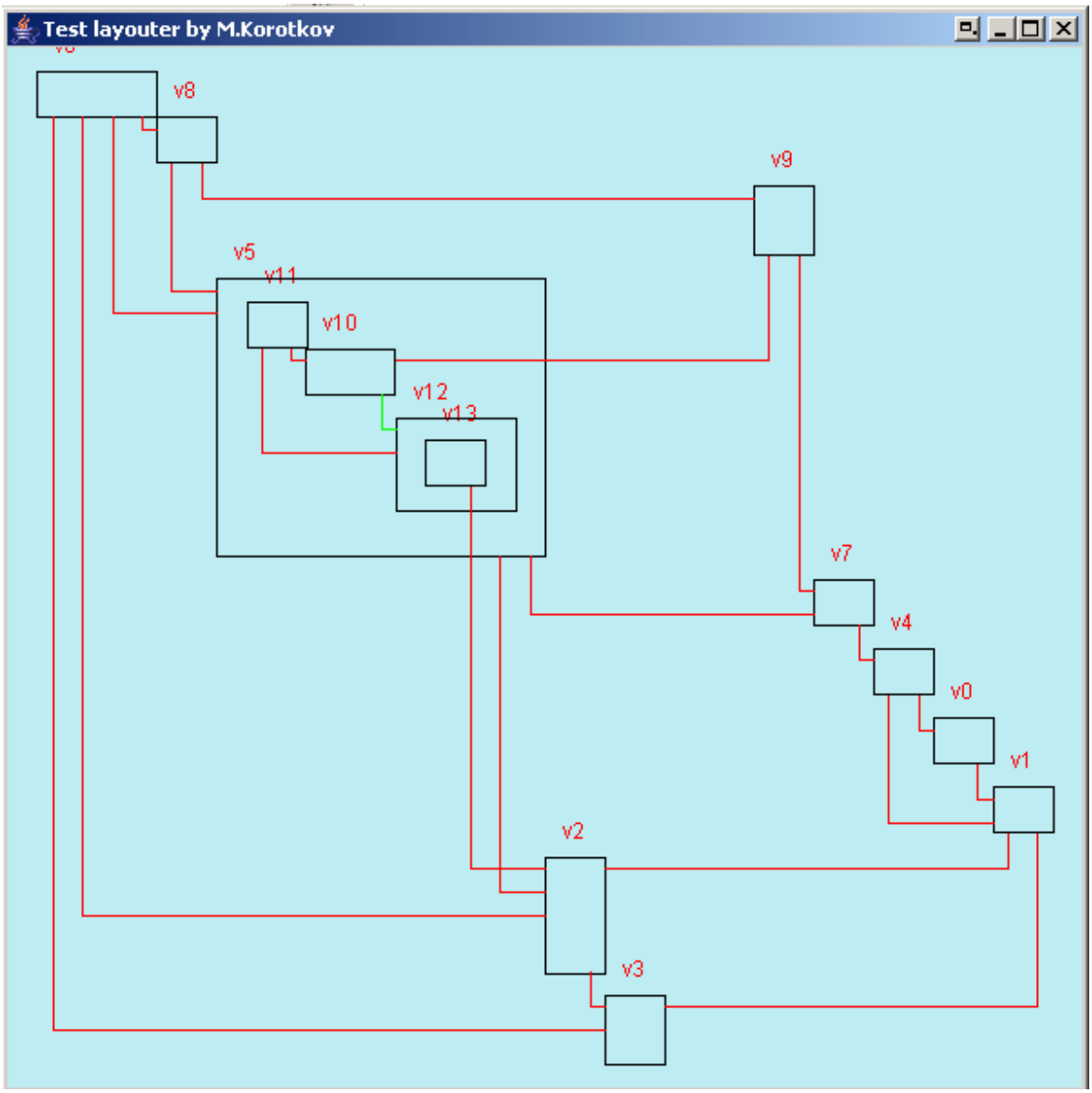

Рис. 40. Укладка со вторым уровнем вложенности.

4.4.4. Внедрение

Алгоритм *QMATH-STATECHART* без повторного использования рядов и колонок был реализован в рамках проекта *UniMod* и входит в *UniMod* версии *1.3*. На рис. 41 приведен пример конечного автомата, уложенного с помощью этого алгоритма.

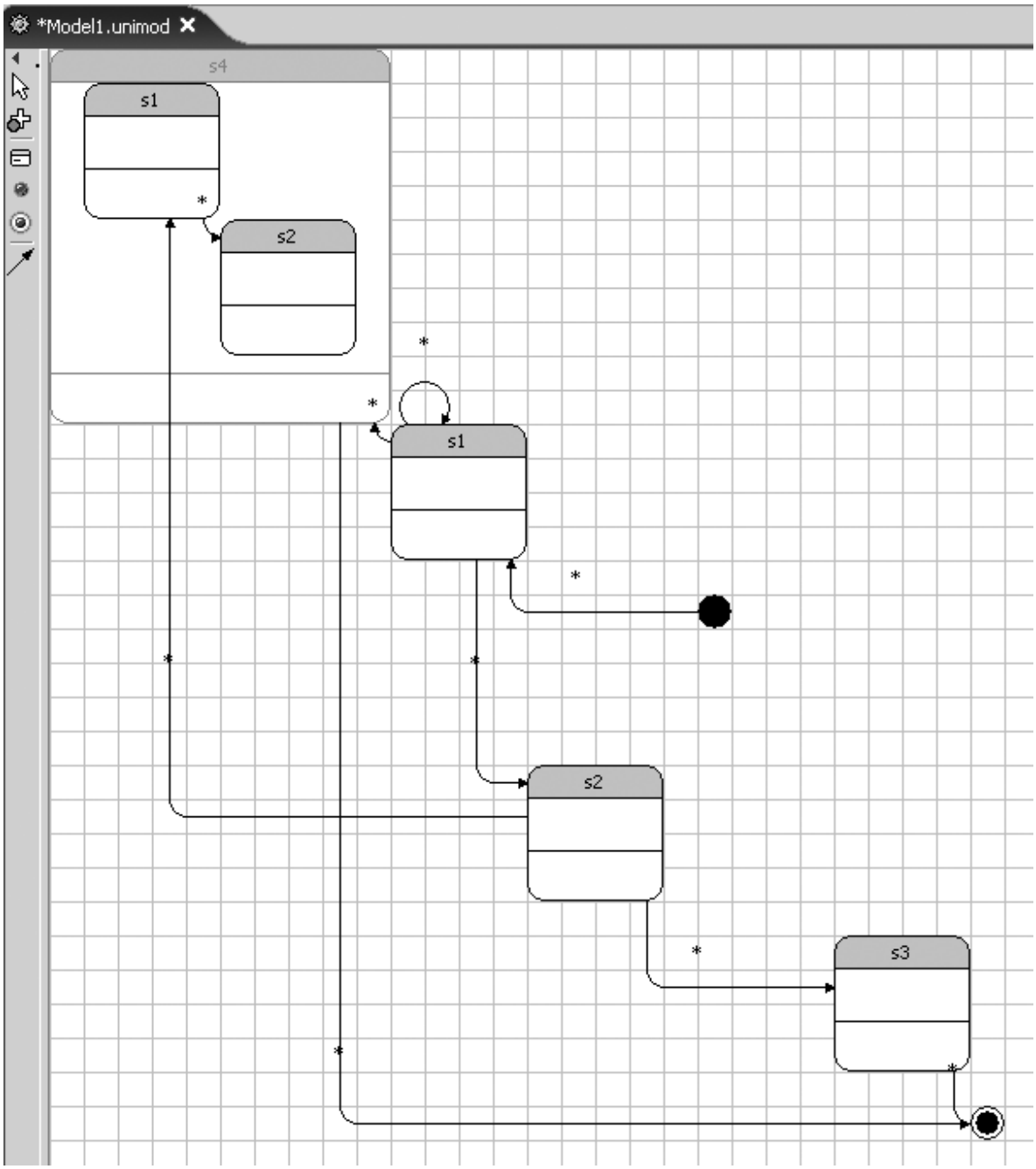

Рис. 41. Результат укладки с помощью алгоритма *QMATH-STATECHART*

Выводы по разделу 4.4

Алгоритм *QMATH-STATECHART* позволяет осуществлять укладку диаграмм состояний. Алгоритм является аналитическим и адаптирован специально для диаграмм состояний. В дальнейшем возможно как улучшение самого алгоритма (уменьшение занимаемой площади, возможно также использование дополнительных методов алгоритма *GIOTTO*), так и дальнейшее улучшение укладки алгоритмами группы методов отжига.

5. СРАВНЕНИЕ АЛГОРИТМОВ

Было произведено сравнение следующих алгоритмов укладки:

1. Метод отжига с ортогонализацией.
2. *QMATH-STATECHART.*
3. Алгоритм *GIOTTO* (для графов общего вида, реализован в пакете *AGD*). Для этого алгоритма рассматривалась только часть диаграмм из тестового пакета.
4. Алгоритм *GIOTTO* (адаптированный для диаграмм состояний, попытка адаптации осуществлена автором).
5. Укладка в *Rational Rose Professional J Edition.*
6. «Ортогональная» укладка в *Together Designer CE.*
7. «Ортогональная» укладка в *NetBeans 5.5 + UML Pack.*

Анализ проводился на тестовом наборе из 30 диаграмм, взятых из коммерческих проектов компании *eVelopers*. Анализ осуществлялся по следующим критериям:

1. Общее качество укладки.

   Субъективная оценка общего качества укладки, производилась автором работы. Такая оценка имеет большое значение, так как задача укладки диаграммы имеет дело, в том числе, и с особенностями эстетического восприятия.
2. Минимизация пересечений.

   Критерий строится по следующим составляющим штрафной функции (смотри описание в разделе 4.1.1):

   - пересечение переходов;

- пересечение вершин;
- прохождение перехода по вершине.

Для всех результатов (все тестовые данные для всех алгоритмов) осуществляется нормирование по десятибалльной шкале, оценка каждого алгоритма вычисляется как среднее его результатов. Затем вычисляется величина 10-X, где X – полученная оценка (дело в том, что максимальное значение штрафной функции соответствует низкому качеству диаграммы, в то время как хочется получить общепринятую десятибалльную шкалу оценок) и округляется до целого числа баллов.

3. Минимизация площади.

   Критерий строится аналогичным образом по критерию компактности диаграммы.

4. Ограничение «свободного места».

   Критерий строится аналогичным образом по следующим составляющим штрафной функции:

   - отклонение от оптимальной длины перехода;
   - отклонение от оптимального расстояния между вершинами.

5. Выравнивание вершин.

   Осуществляет ли алгоритм выравнивание вершин по крупной сетке.

6. Унификация размеров простых состояний.

   Осуществляет ли алгоритм унификацию размеров простых состояний.

7. Ортогонализация укладки.

Является ли построенная укладка ортогональной (заметим, что даже «ортогональные» алгоритмы укладки в существующих продуктах в действительности строят неортогональную укладку для некоторых ребер).

8. Быстродействие (до 20 состояний).

   Время построения укладки на малых диаграммах. Для всех решений учитывалось, в том числе, время, затраченное на перерисовку диаграммы, поскольку в коммерческих решениях не представляется возможным выделить собственно время на укладку диаграммы. В целом, для разработанных алгоритмов известны оценки вычислительной сложности, а задача точной оценки быстродействия сторонних решений не ставилась, поэтому в тестах быстродействия фиксировалось только «ощутимое» время задержки при укладке диаграммы. В случае если укладка строилась без задержек, ставилась оценка «10».

9. Быстродействие (до 100 состояний).

   Время построения укладки на диаграммах среднего размера.

Ниже приведена сравнительная таблица характеристик для выбранных алгоритмов.

Таблица 5. Сравнительные характеристики

Характеристика	**1**	**2**	**3**	**4**	**5**	**6**	**7**
Общее качество укладки	7	7	10	6	3	5	4
Минимизация пересечений	6	7	8	7	2	4	6
Минимизация площади	7	6	9	5	5	6	8
Ограничение «свободного места»	7	8	8	8	8	7	6

Выравнивание вершин	есть	есть	есть	есть	нет	есть	нет
Унификация размеров простых состояний	есть	есть	есть	есть	есть	есть	есть
Ортогонализация укладки	есть	есть	есть	есть	нет	нет	нет
Быстродействие (до 20 состояний)	8	10	10	10	10	10	10
Быстродействие (до 100 состояний)	4	9	10	10	10	9	9

Таким образом, можно утверждать, что:

1. Все рассмотренные редакторы *UML* осуществляют укладку низкого качества. К примеру, на рис. 42 приведена укладка диаграммы состояний с помощью алгоритма ортогональной укладки в *Together Designer CE* (та же диаграмма уложена на рис. 40 с помощью алгоритма *QMATH-STATECHART*). Качество укладки значительно хуже, как минимум по следующим критериям:
    - есть немотивированные прохождения ребер по состояниям;
    - очень большое количество пересечений;
    - нарушается ограничение «свободного места»;
    - для некоторых ребер нарушается ортогональность.

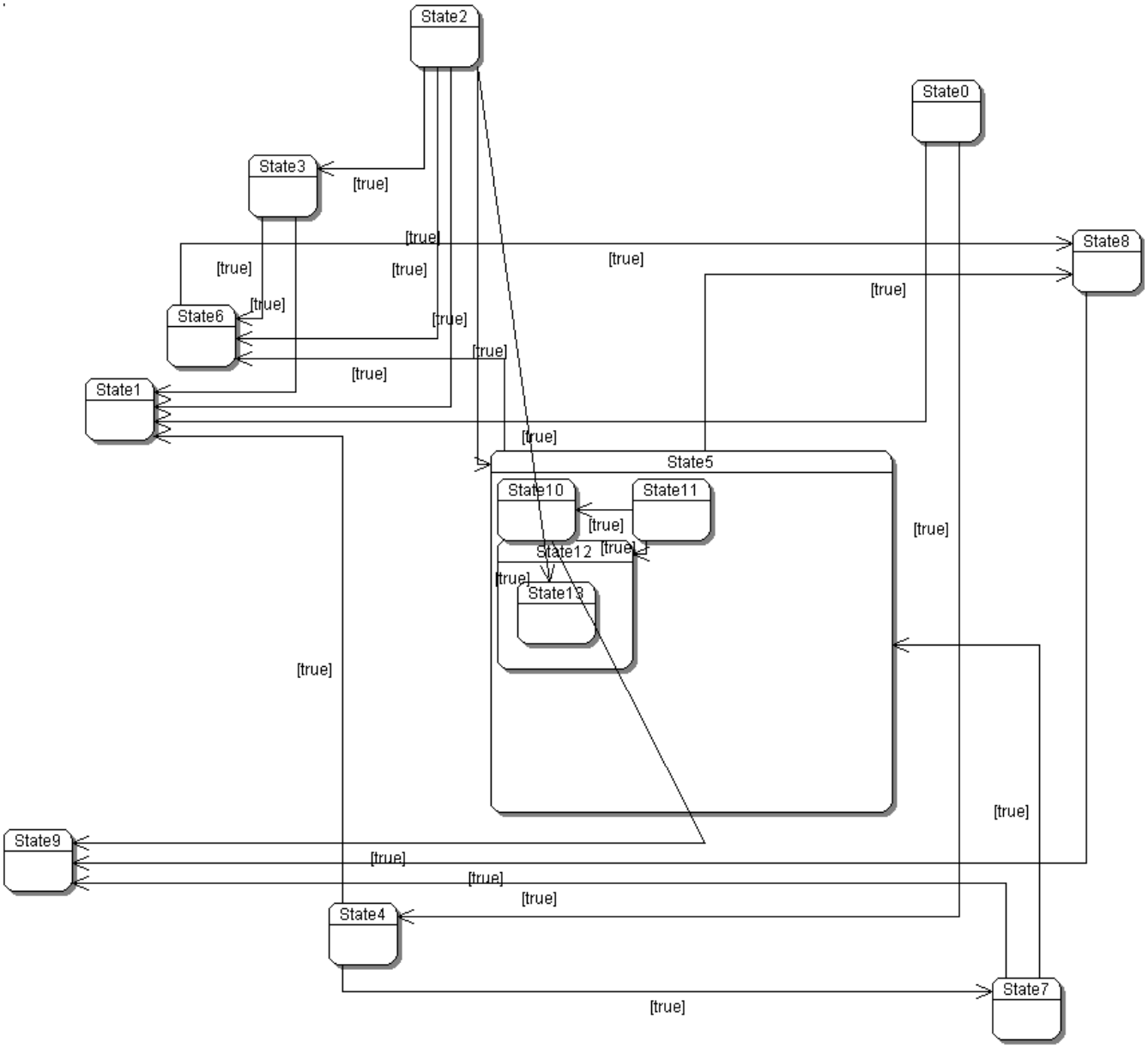

Рис. 42. Для сравнения: *Together Designer CE*

2. Графы общего вида (в том числе и с вершинами переменного размера) лучше всего укладывает алгоритм *GIOTTO*. Однако, как подробнее описано в разд. 4.2, автору не удалось ни найти, ни построить адаптацию этого алгоритма к укладке диаграмм состояний приемлемого качества.

3. Из разработанных автором алгоритмов, метод отжига с ортогонализацией лучше подходит для небольших диаграмм, а алгоритм *QMATH-STATECHART* – для диаграмм среднего размера. Оба алгоритма превосходят по качеству укладки алгоритмы, применяемые в рассмотренных продуктах.

ЗАКЛЮЧЕНИЕ

В настоящей работе, являющейся развитием работы [9], исследована проблема укладки диаграмм состояний *UML*. Произведено исследование существующих продуктов, как предназначенных для редактирования диаграмм и предоставляющих возможность укладки, так и специализирующихся на укладке графов (разд. 2). Результаты исследования показали, что задача укладки диаграмм состояний является актуальной и практически значимой. Ее решение может найти применение в целом ряде программных продуктов.

Произведен подробный анализ задачи, выделен ряд эстетик, базируясь на которых проводится оценка качества укладки (разд. 1). Дан обзор алгоритмов укладки (разд. 1.3, 3.2).

Осуществлено дальнейшее развитие метода отжига с ортогонализацией (разд. 4.1). Разработаны новые аналитические алгоритмы укладки – алгоритмы *QMATH* (разд. 4.3) и специально адаптированный для укладки диаграмм состояний *QMATH-STATECHART* (разд. 4.4). Намечены пути дальнейшего развития алгоритма. Алгоритм *QMATH-STATECHART* реализован в продукте *UniMod.*

Произведено сравнение алгоритма *QMATH-STATECHART* с методом отжига с ортогонализацией, алгоритмом *GIOTTO,* а также с существующими решениями, реализованными в коммерческих продуктах, поддерживающих укладку диаграмм (разд. 5).

ИСТОЧНИКИ

1. *Гуров В.С., Мазин М.А.* Веб-сайт проекта *UniMod*. http://unimod.sourceforge.net/
2. *Шалыто А.А., Туккель Н.И.* Танки и автоматы // BYTE/Россия. 2003. № 2, с. 69–73. http://is.ifmo.ru/ (раздел «Статьи»).
3. *Rumbaugh J., Jacobson I., Booch G.* The Unified Modelling Language Reference Manual. Second Edition. MA: Addison-Wesley, 2004.
4. *Шалыто А.А.* Switch-технология. Алгоритмизация и программирование задач логического управления. СПб.: Наука, 1998.
5. *Новиков Ф.А.* Дискретная математика для программистов. СПб.: Питер, 2004.
6. *Tamassia R.* Advances in the Theory and Practice of Graph Drawing. http://www.cs.brown.edu/people/rt/papers/ordal96/ordal96.html
7. *Касьянов В.Н., Евстигнеев В.А.* Графы в программировании: обработка, визуализация, применение. СПб.: БХВ-Петербург, 2003.
8. *Battista G., Eades P., Tamassia R., Tollis I.* Graph Drawing. Algorithms for the Visualization of Graphs. New Jersey: Prentice Hall, 1999.
9. *Коротков М.* Разработка и реализация алгоритма укладки диаграмм состояний. СПбГУ ИТМО. Бакалаврская диссертация. 2005. http://is.ifmo.ru/papers/glayout/
10. *Sugiyama K.* Graph Drawing and Applications for Software and Knowledge Engineers. Singapore: Mainland Press, 2002.
11. *Frankel D.* Model Driven Architecture. Applying MDA to Enterprise Computing. Indianapolis: Web Publishing Inc., 2003.
12. *Frutcherman T., Reingold E.* Graph Drawing by Force-directed Placement // Software – Practice and Experience, 1991, vol. 21, pp. 1129–1164.

13. *Makinen E., Seiranta M.* Genetic algorithms for drawing bipartite graphs // International Journal of Computer Mathematics, 1994, vol. 53, No 3, pp. 157 – 166.
14. Веб-сайт проекта AGD (Algorithms for Graph Drawing). http://www.ads.tuwien.ac.at/AGD/
15. Веб-сайт проекта GDT (Graph Drawing Toolkit). http://www.dia.uniroma3.it/~gdt
16. Веб-сайт проекта Graph Layout Toolkit компании Tom Sawyer Software http://www.tomsawyer.com/tsl/tsl.java.php
17. *Battista G., Garg A., Liotta G., Tamassia R., Tassinari E., Vargiu F.* An experimental comparison of four graph drawing algorithms // Computational Geometry, 1997, 7(5–6), pp. 303–325.
18. *Tamassia R., Battista G., Batini C.* Automatic graph drawing and readability of diagrams // IEEE Transactions on Systems Man Cybernetics, 1998, 18(1), pp. 61–79.
19. *Klaw G., Mutzel P.* Automatic layout and labeling of state diagrams / Materials of Graph Drawing conference, 1997.
20. *Степанян К.* Автоматическое представление модели UML. СПбГПУ. Магистерская диссертация. 2003.
21. *Кормен Т., Лейзерсон Ч., Ривест Р.* Алгоритмы: построение и анализ. М.: МЦНМО, 1999.
22. *Hsu T., Ramachandran V.* On finding a smallest augmentation to biconnect a graph // SIAM Journal on Computing, 1993, 22, pp. 889–912.
23. *Ueno S., Kajitani Y., Wada H.* Minimum augmentation of a tree to a k-edge connected graph // Networks, 1988, 18, pp. 19–25.
24. *Harris J.* JGraphEd – A Java Graph Editor and Graph Drawing Framework. Carleton University, School of computer science. 2004.

25. *Maon Y., Schieber B., Vishkin U.* Parallel ear decomposition search and st-numbering in graphs. Courant Institute of Mathematical Sciences / Ultracomputer note № 102, Computer Science Department Technical Report № 222, 1986.

26. *Klein M.* A Primal Method for Minimal Cost Flows with Application to the Assignment and Transportation Problems //Management Science, 1967, 14, pp. 205–220.

27. *Papakostas A.*, *Tollis I.G.* Algorithms for area-efficient orthogonal drawings. // Computational Geometry, 1998, 9(1–2), pp. 83–110.

28. *Biedl T.*, *Kant G.* A Better Heuristic for Orthogonal Graph Drawings. Vol. 8500 of Lecture Notes Computer Science, pp. 24–35. Springer–Verlag, 1994.